Lots of different RT systems

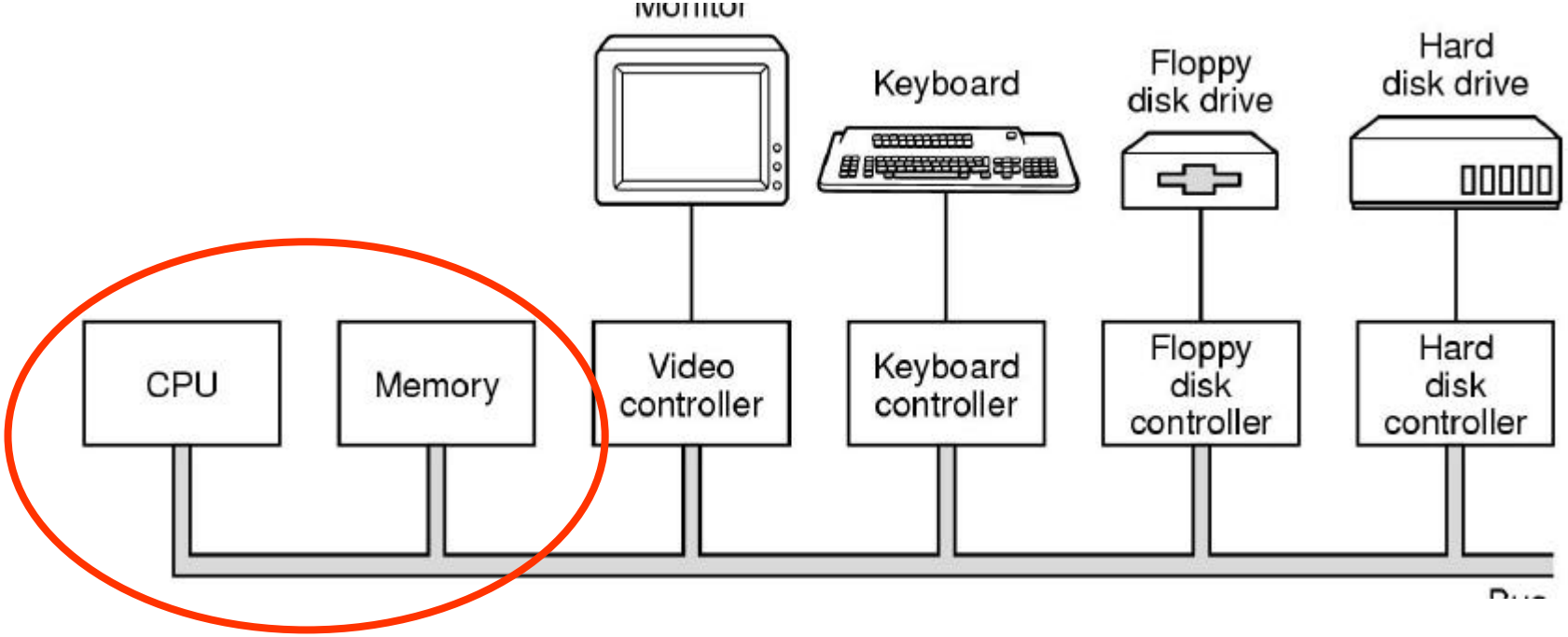
	operating system	programming language	application field
? signal processors	none	assembler, C, CHILL	fast systems
? micro controllers / microprocessors	none	assembler, C	system appl.
? PLC *)		many	industry
? PC - stations	DOS, Windows	many, graphic lang.	user interface
? dedicated card systems	OS9, VRTX	C, C++	larger decentr. systems
? workstations	UNIX	ADA, C, C++	PPS *), CIM
	VMS	Cobol, Fortran, Pascal	

*) PLC programmable logic controller
PPS production planning system

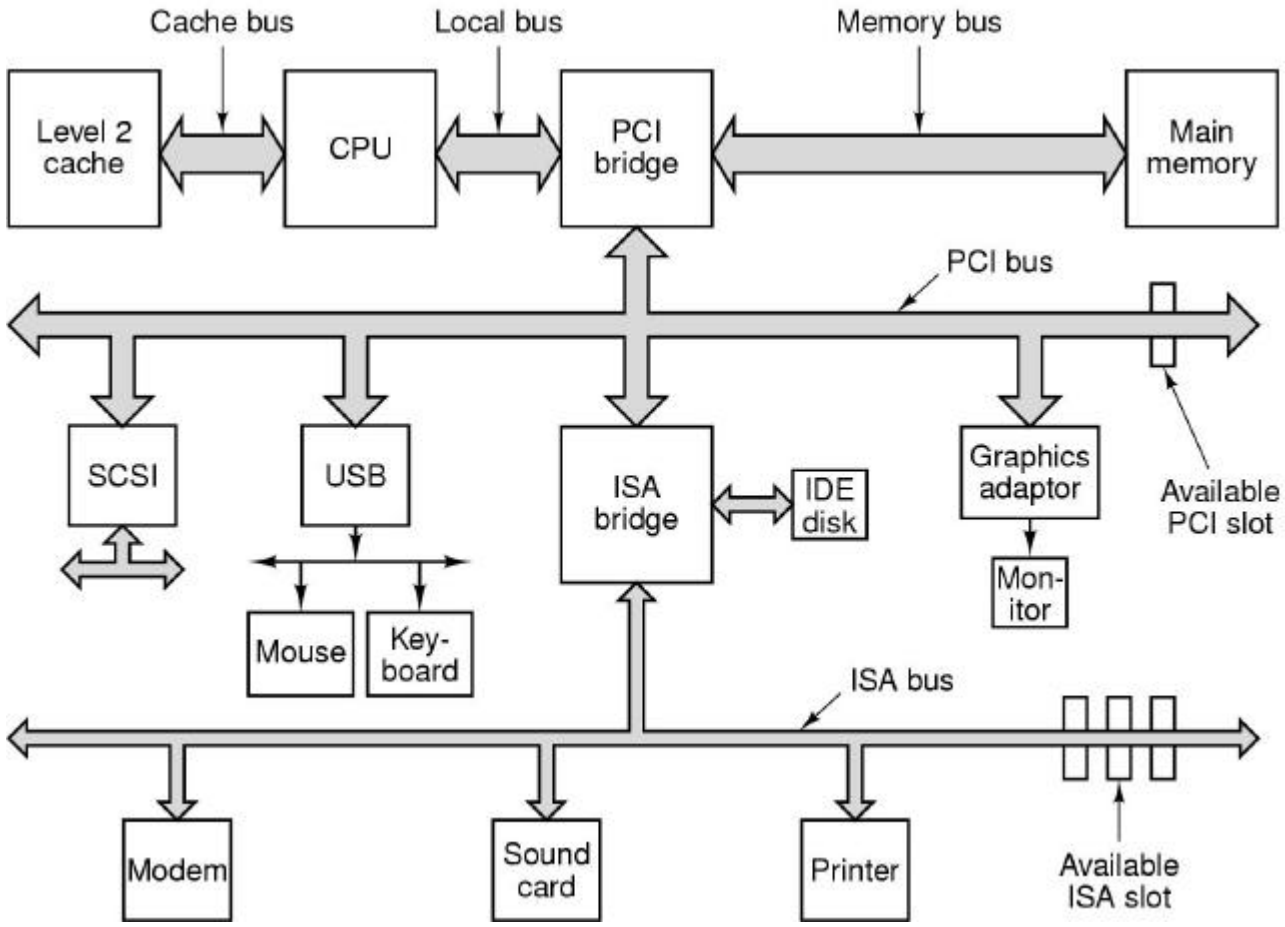
Solving of RT problems

- with a standard processor (e.g. Intel x86 or Motorola 680x0) or
- with a specialised processor or
- with an updated standard processor (e.g. enhanced periphery)

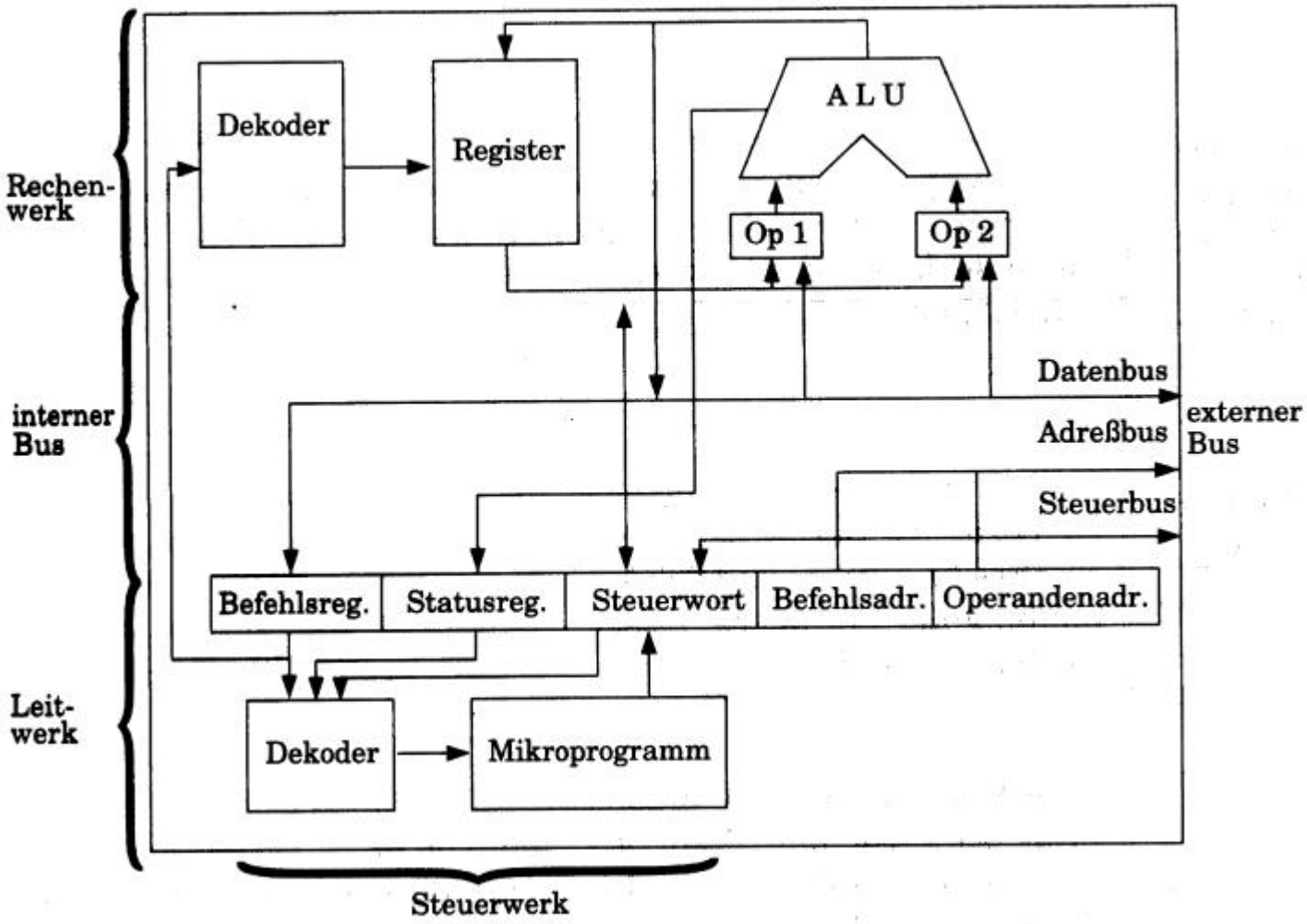
Lots of different RT systems - all with the CPU? memory dualism



CPU? memory dualism and more than one bus

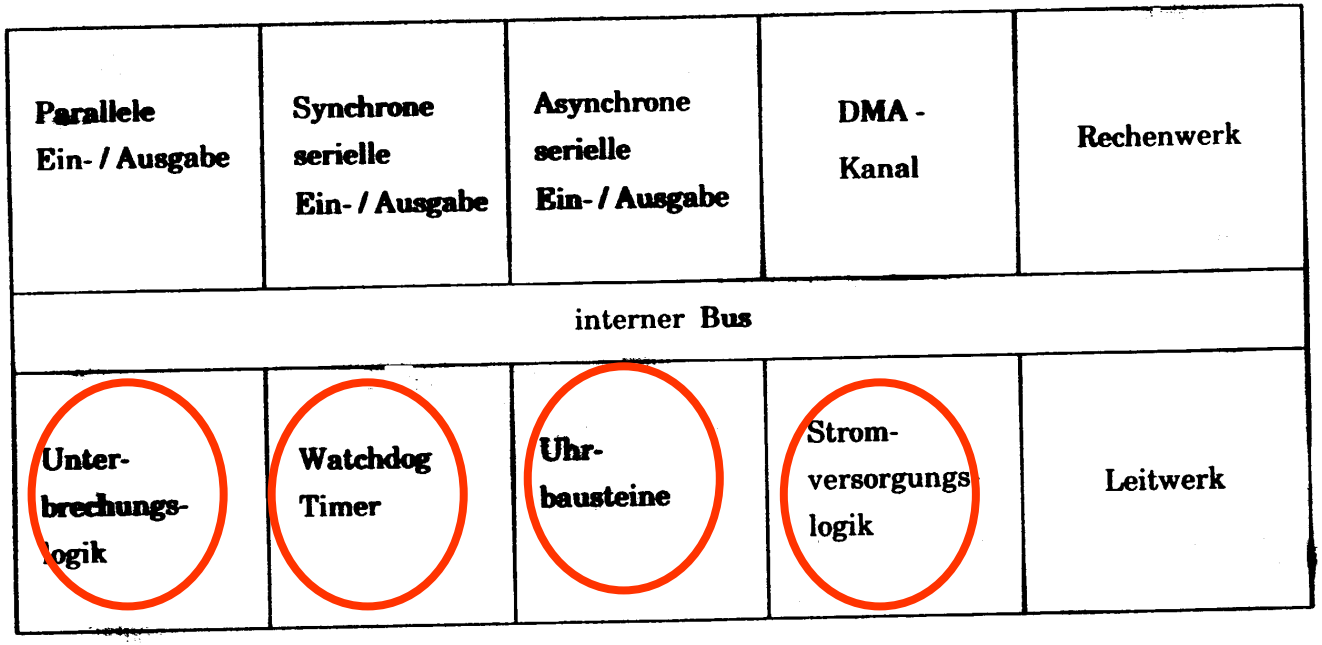


CPU internal

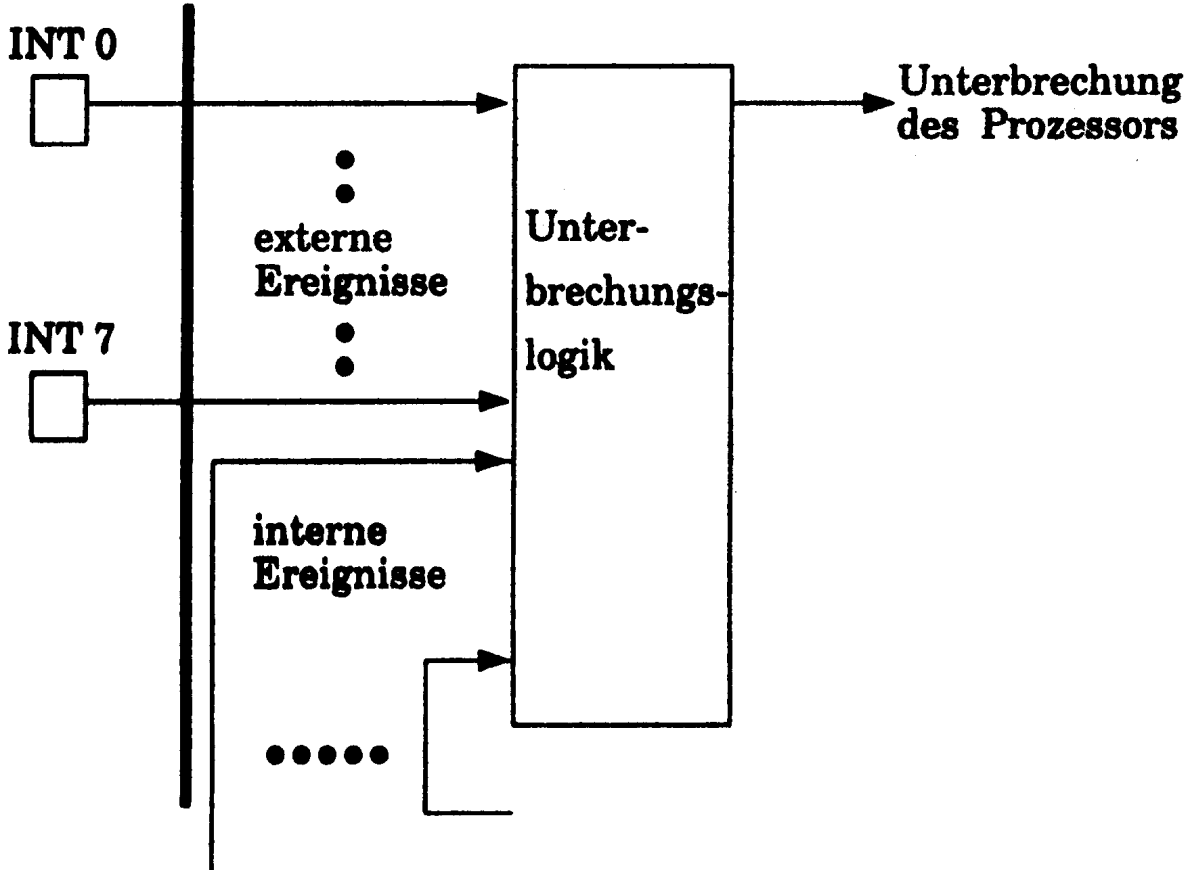


Internal architecture of a standard processor

Exkurs – RT-Anpassungen für Standard-Prozessoren

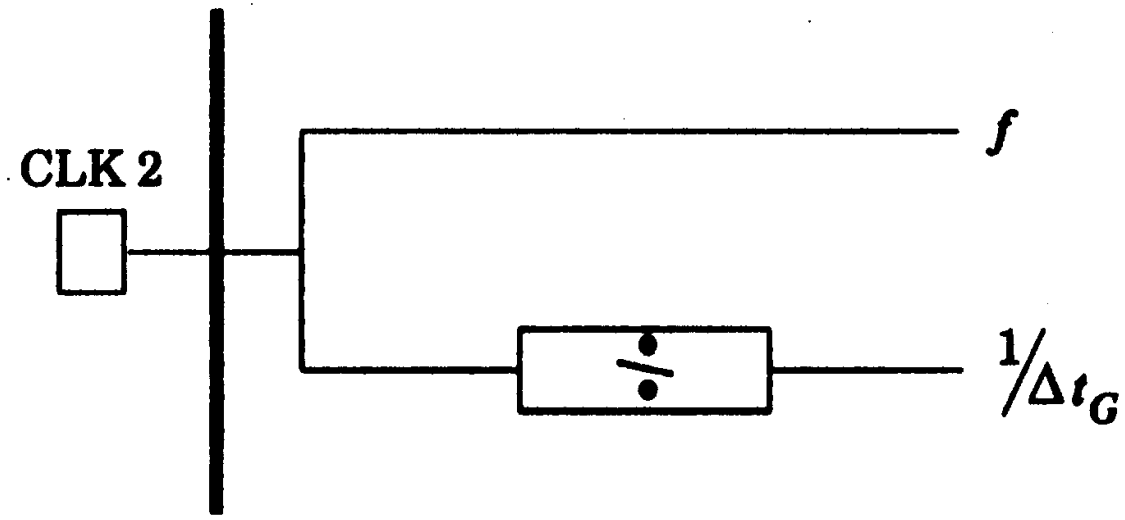


RT-Anpassung: Unterbrechungslogik

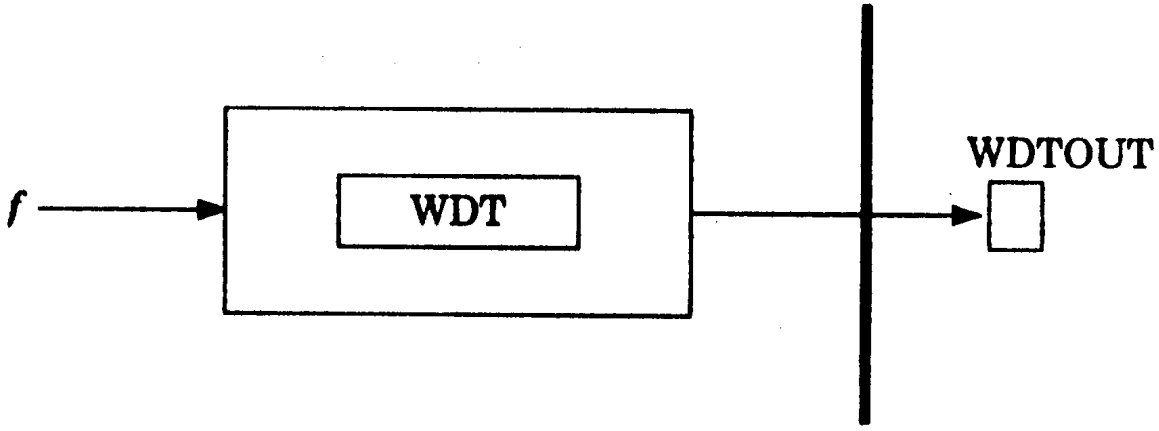


Interne und externe Eingänge der Unterbrechungslogik

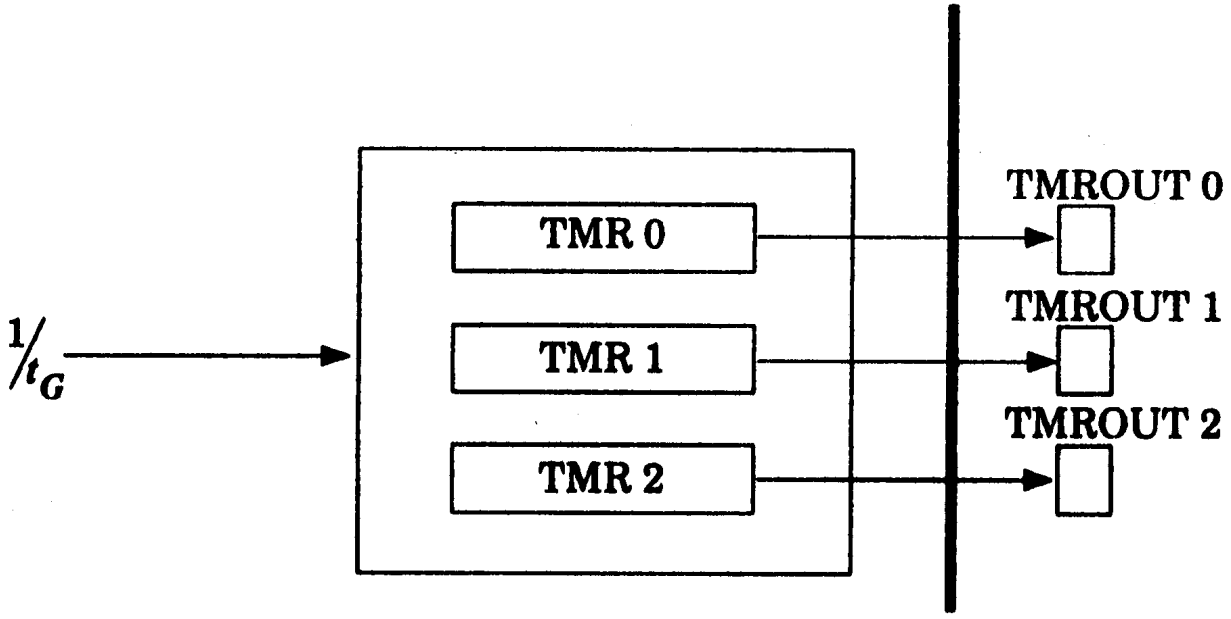
RT-Anpassung: Zeitsystem



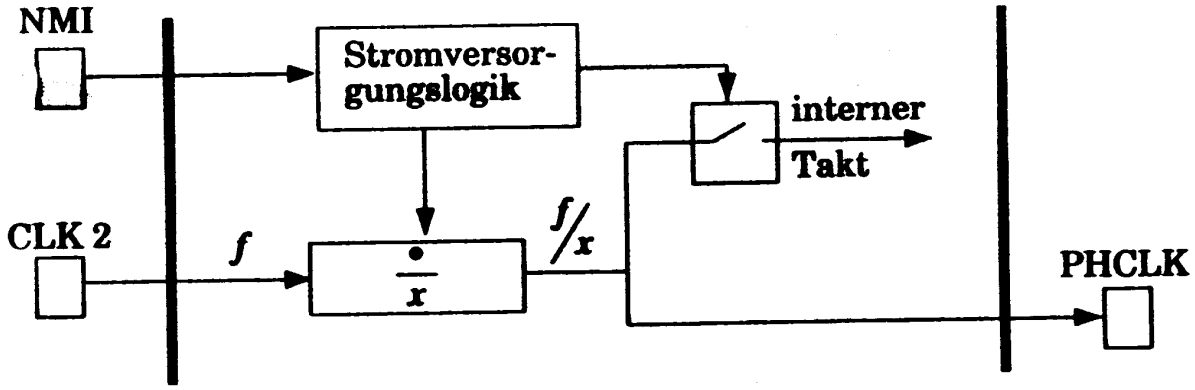
RT-Anpassung: Watchdog Timer



RT-Anpassung: Uhrbaustein mit Zählern



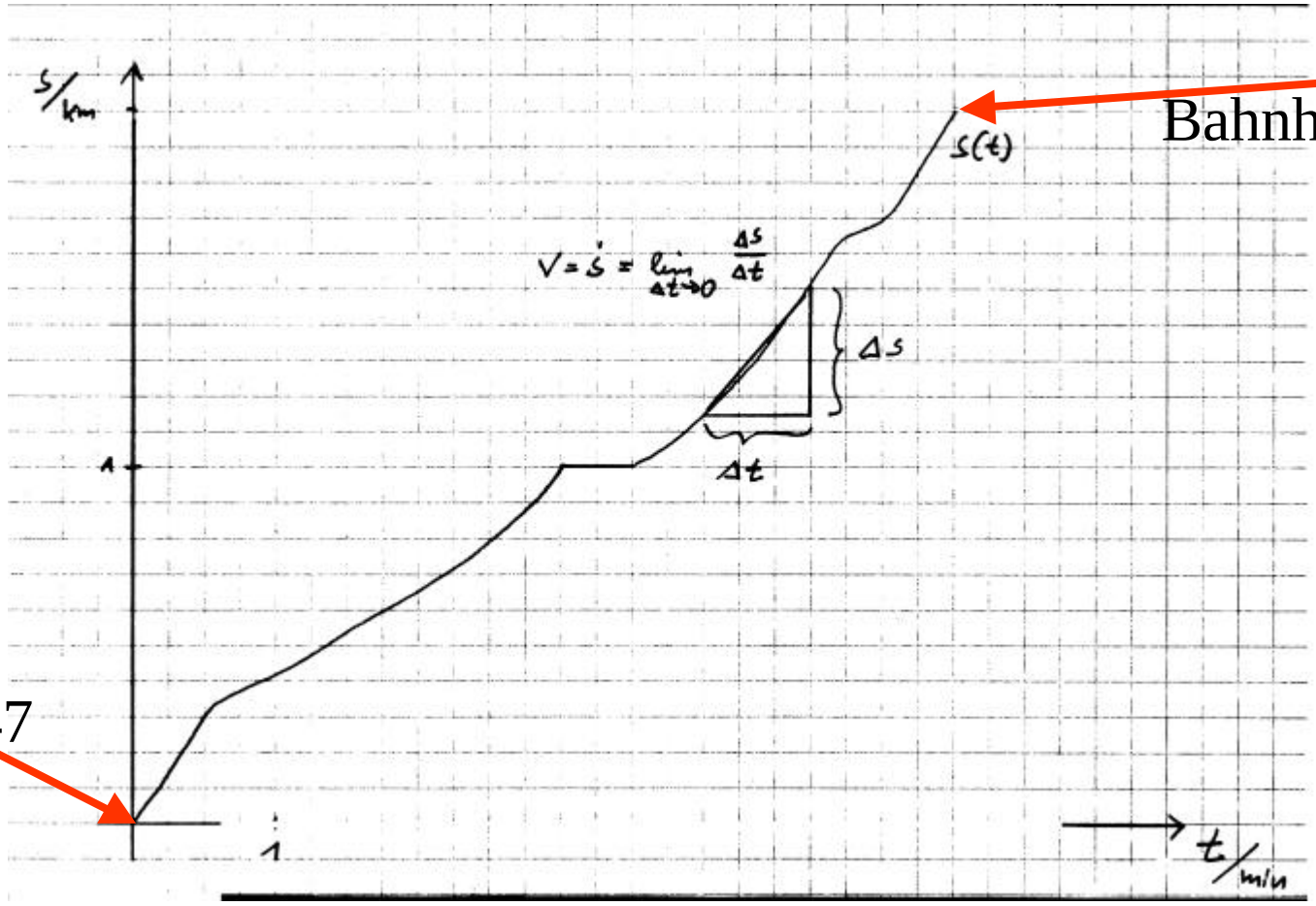
RT-Anpassung: Strom(spar)versorgungslogik



Reduktion der Taktrate zur Stromeinsparung

Polling and interrupt handling

Task: count impulses within a period of time and calculate speed

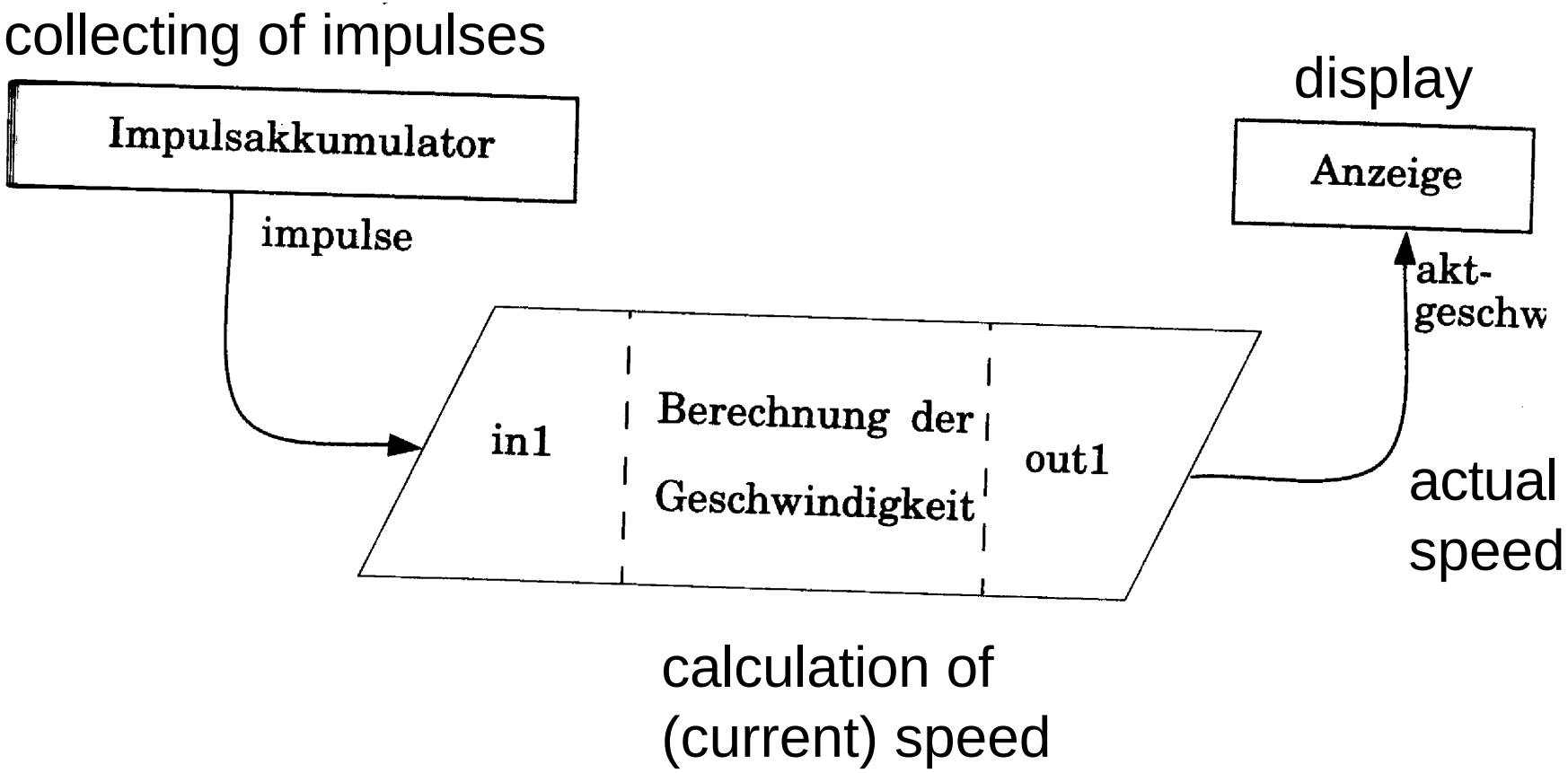


Bahnhofstr. 61-63

An der
Weberei 47

Polling and interrupt handling

Task: count impulses within a period of time and calculate speed

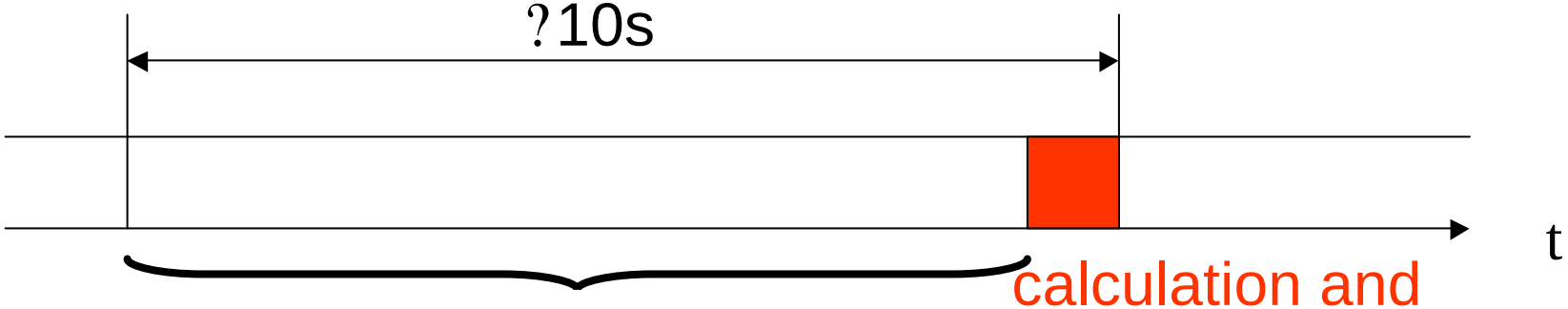


Polling and interrupt handling

```
#define TRUE 1
#define INDEX ...
#define DELTAP 10           /* Periode: 10s */
#define PERIMETER 2.232     /* Perimeter of 28# wheel in m */
#define DIMKMH 3600/1000    /* from m/s to km/h */

int impulse;                /* measured */
int preimpulse = 0;         /* for reinitialization */
double actdistance;         /* in m */

main ()
{
   while (TRUE) {
      cycle (INDEX);
      in1 (&impulse);
      actdistance = (impulse - preimpulse) * PERIMETER;
      preimpulse = impulse;
      out1 ((actdistance/DELTAP) * DIMKMH);
   }
}
```

Polling and interrupt handling

```
#define TRUE 1
#define INDEX ...
#define DELTAP 10                          /* Periode: 10s */
#define PERIMETER 2.232       /* Perimeter of 28# wheel in m */
#define DIMKMH 3600/1000      /* from m/s to km/h */

int impulse;                               /* measured */
int preimpulse = 0;                        /* for reinitialization */
double actdistance;                        /* in m */

main ()
{
       while (TRUE) {
              cycle (INDEX);
              in1 (&impulse);
              actdistance = (impulse - preimpulse) * PERIMETER;
              preimpulse = impulse;
              out1 ((actdistance/DELTAP) * DIMKMH);
       }
}
```



active wait in function `cycle` depending from `INDEX`

calculation and display of speed

Problems with that Polling

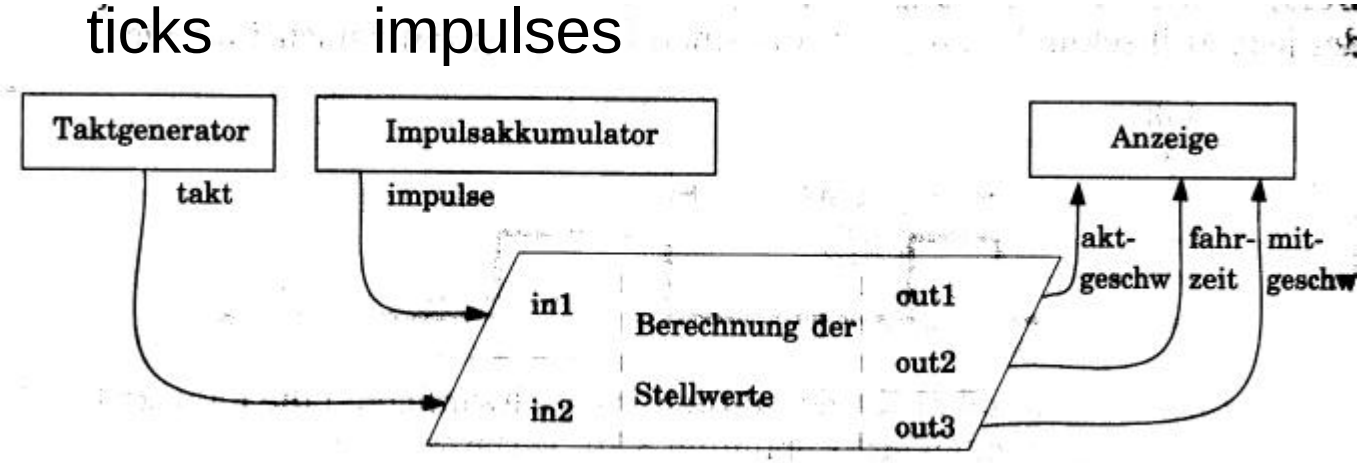
real cycle time differs from used cycle time - this may cause unlimited timing errors on accumulated timing values (remember Patriot example!)

system is not stable ("robust") - any change of calculation or HW (i.e. new processor, other tact rate, other periphery) causes changes (e.g. in procedure `cycle (INDEX);`)

if there are more than one timing constraints in a RTS or is the timing of the calculation data dependent the polling strategy becomes unclear/confusing and faults may (will!) happen

Polling with clock

System is enhanced by
- clock generator giving 1 tick/ms
- display of driving time and average speed



Polling with clock(1)

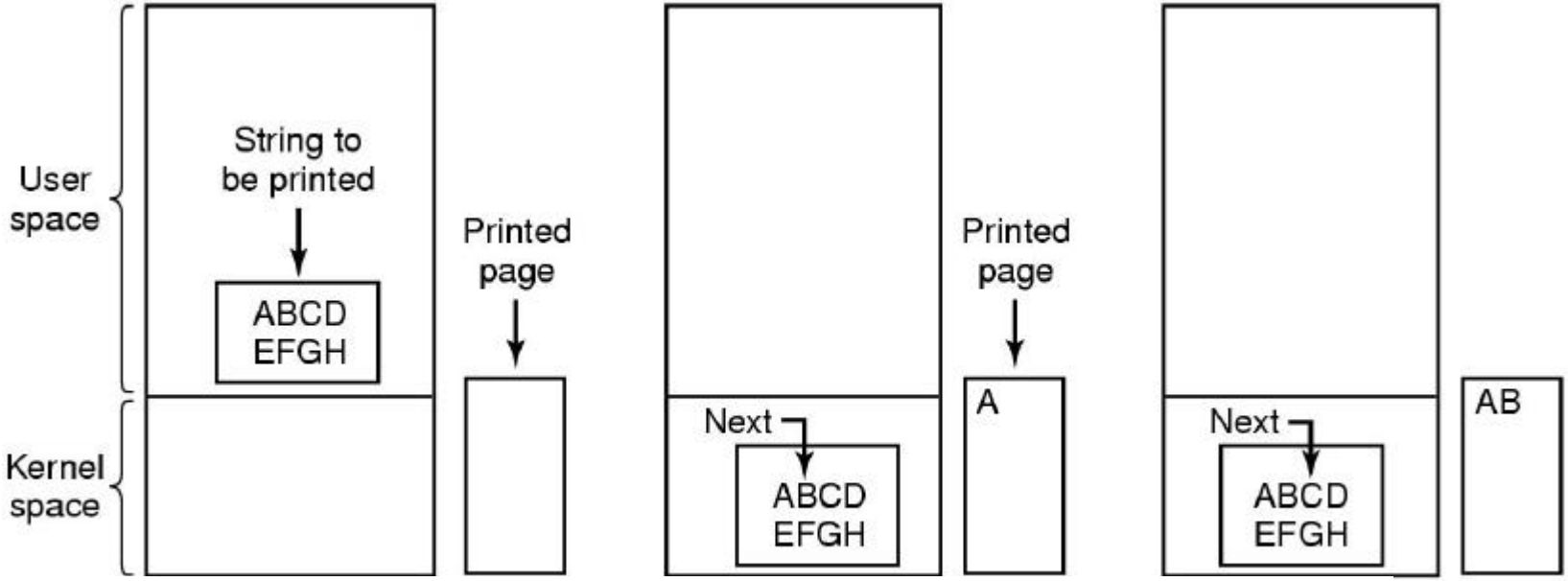
```
#define TRUE 1
#define INDEX ...
#define DELTAP 10           /* Periode: 10s */
#define PERIMETER 2.232     /* Perimeter of 28# wheel in m */
#define DIMKMH 3600/1000    /* from m/s to km/h */
#define tG 0.001            /* clock gives ms ticks */
```

Polling with clock(2)

```
main ()
{
    int tick, impulse;          /* measured */
    int j = 0;                  /* counter of ticks of a periode */
    int i = 0;                  /* counter of periodes */
    int drivingtime = 0;        /* driving times in periods */
    int preimpulse = 0;
    double actdistance, totdistance;            /* in m */
    while (TRUE) {
        do in2(&tick); while (tick == 0); /* instead of cycle(INDEX); */
        if (++j * tG >= DELTAP) {
            j = 0;      /* end of period */
            drivingtime = ++I * DELTAP;
            in1 (&impulse);
            actdistance = (impulse - preimpulse) * PERIMETER;
            totdistance = impulse * PERIMETER;
            preimpulse = impulse;
            out1 ((actdistance/DELTAP) * DIMKMH);
            out2 ( drivingtime );
            out3 ((totdistance/drivingtime) * DIMKMH);
        }
        do in2(&tick); while (tick == 1);
    }
}
```

Another polling example: print of a string

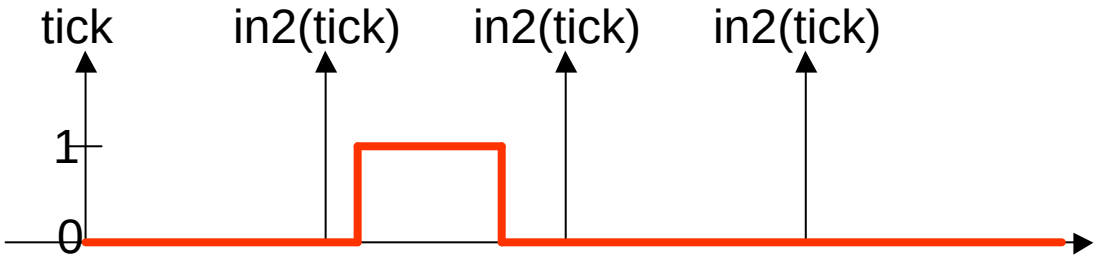
Steps for printing a string



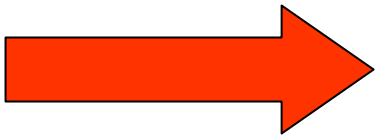

```
copy_from_user(buffer, p, count);                  /* p is the kernel bufer */
for (i = 0; i < count; i++) {                      /* loop on every character */
        while (*printer_status_reg != READY) ;     /* loop until ready */
        *printer_data_register = p[i];             /* output one character */
}
return_to_user( );
```

Remaining problems with polling

two subsequent viewing checks may miss a change of a value in between:



most of the processor time is used to check values which didn´t change, a terrible waste of ressources

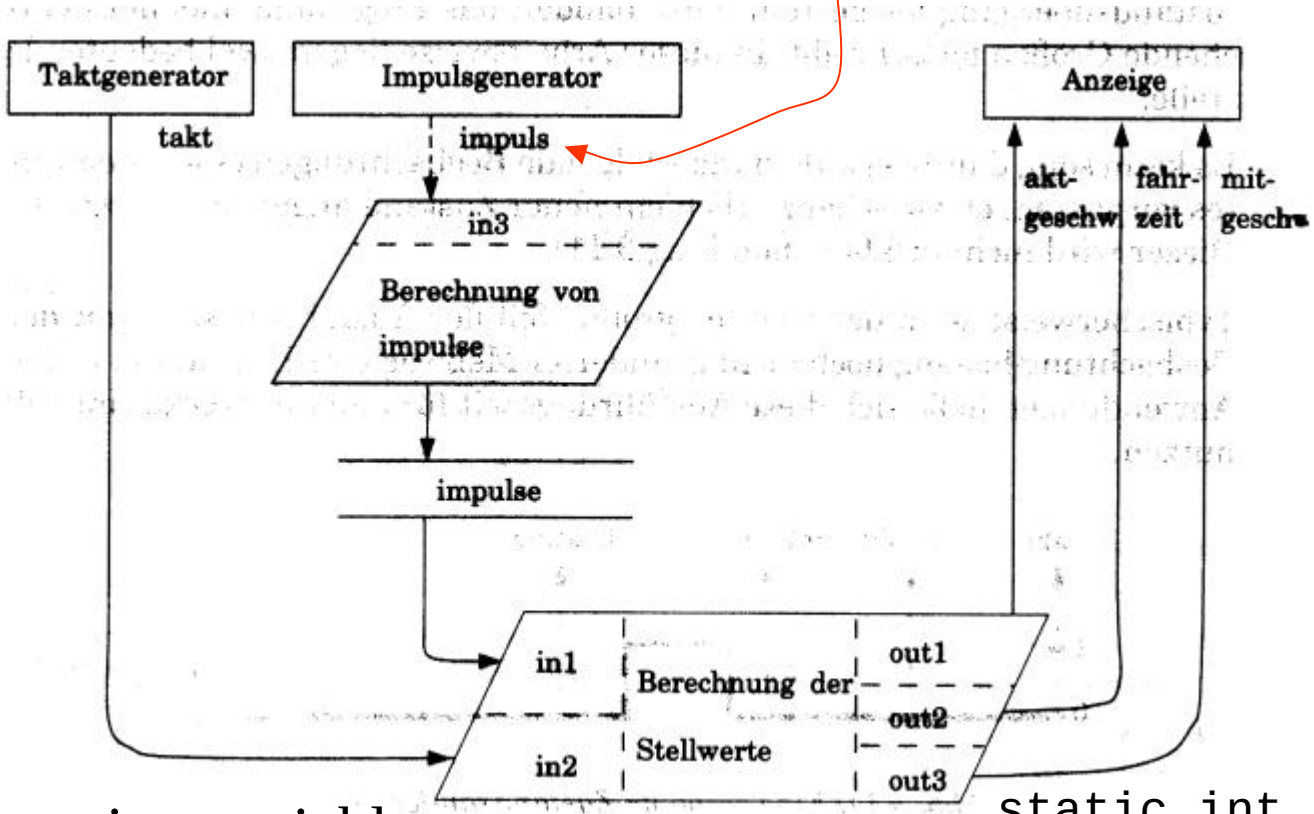


interrupt handling:

- bundle all activities necessary at a state change to a process
- start this process at state change

Polling vs interrupt handling

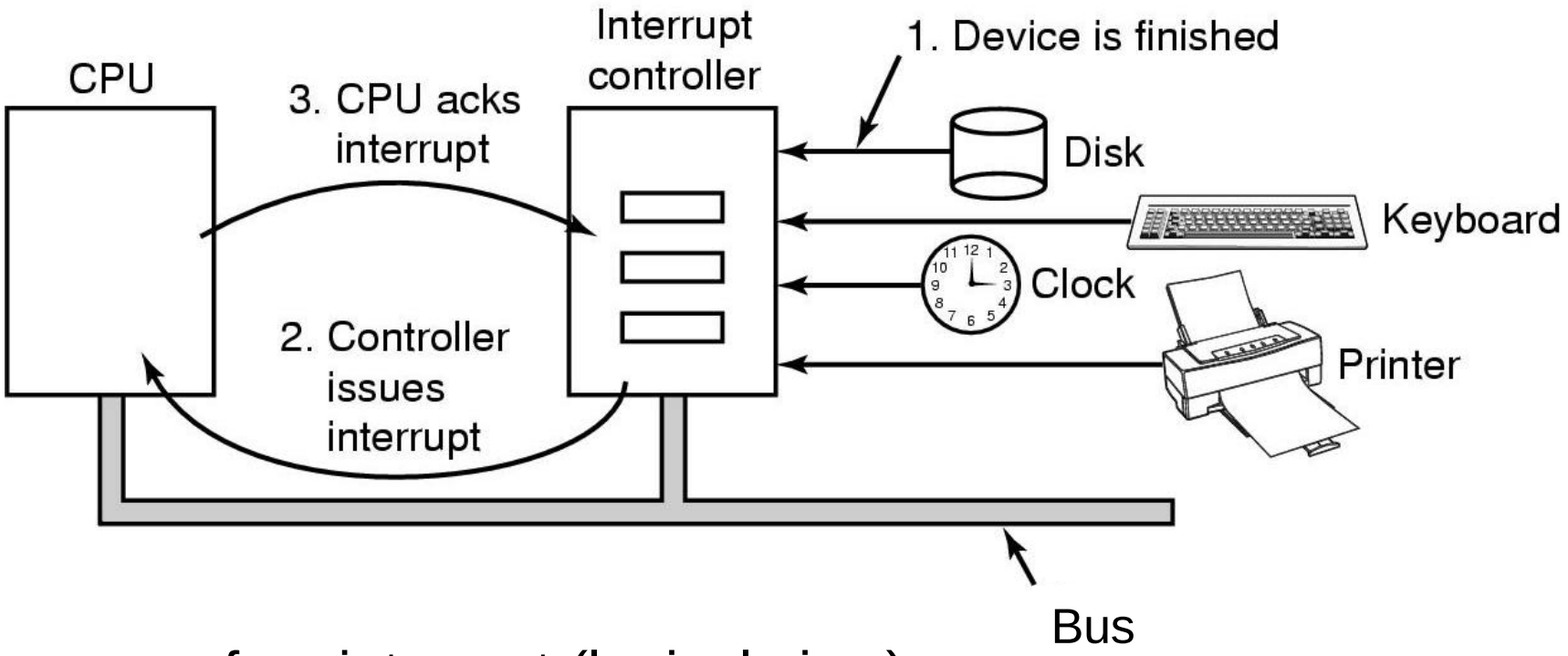
measure system now deliveres only single impuls (not accumulated ones)



where `impulse` is a variable of the interrupt process, read with `in1` by `main()`

```
static int impulse = 0;
in3()
{
    impulse++;
}
```

Steps within a interrupt

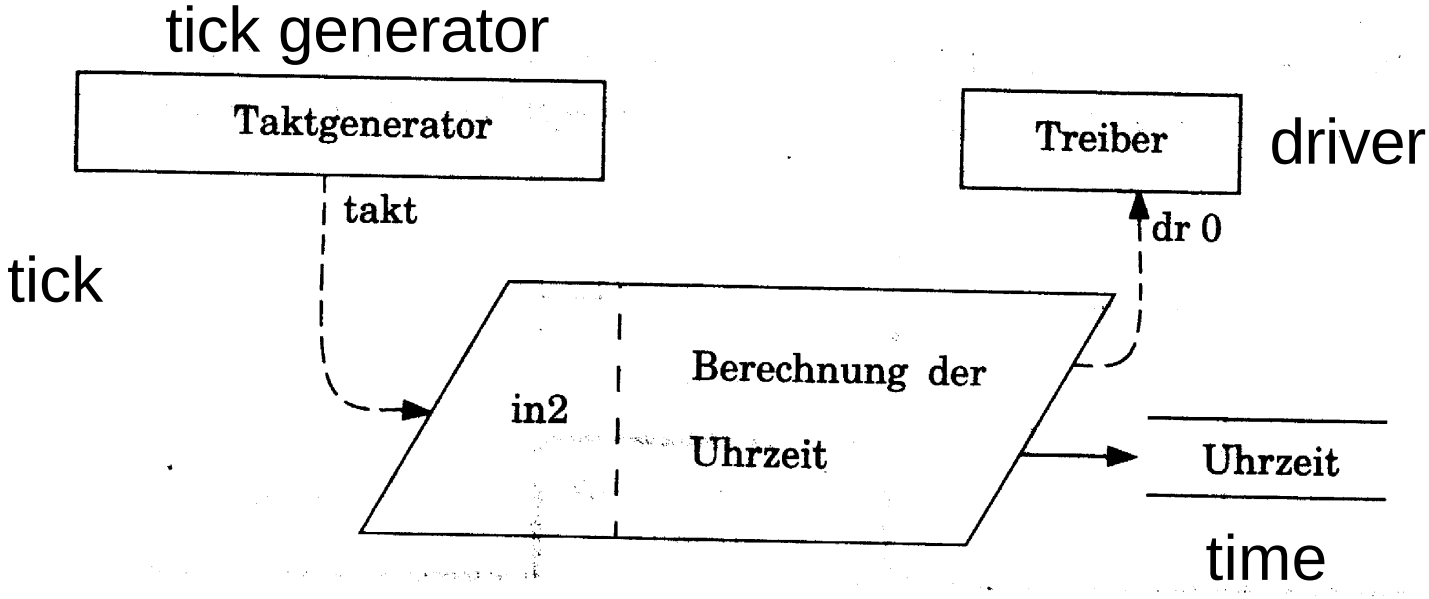


Sequence of an interrupt (logical view):
The connections between device and interrupt controller are using interrupt lines on the bus instead of own lines.

Polling vs interrupt handling

Interrupt processes plus drivers build lowest SW level for interface periphery.

Example:
interrupt process for calculation of time is triggered by tick



Polling vs interrupt handling

```
#define tGrt 1000   /* 1/deltatG granularity */
type def struct {
int s, m, h;
} time;
static time t = {0, 0, 0};
in2();              /* interrupt process */
{ static int ticks = 0;
   if (++ticks == tGrt) {                /* second */
      ticks = 0;
      if (++t.s == 60) {                 /* minute */
         t.s = 0;
         if (++t.m == 60) {              /* hour */
            t.m = 0;
            wakeup (dr0)
            if (++t.h == 24) {           /* day */
               t.h = 0;
            }
         }
      }
   }
}
```

Another example (print of string) again: now with interrupt

Output of string on printer via "interrupt driven I/O":

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

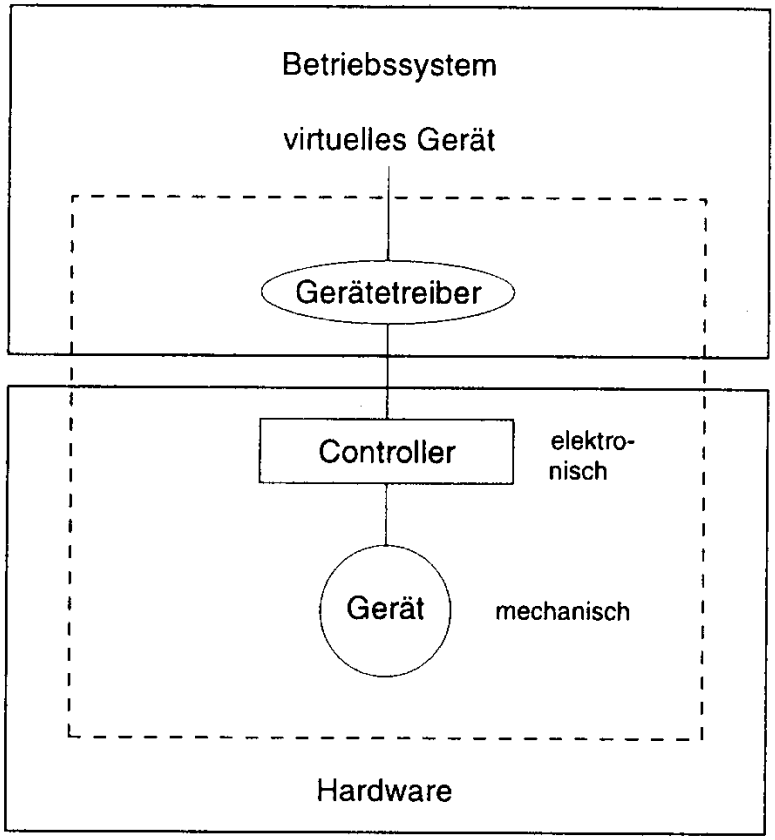
```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

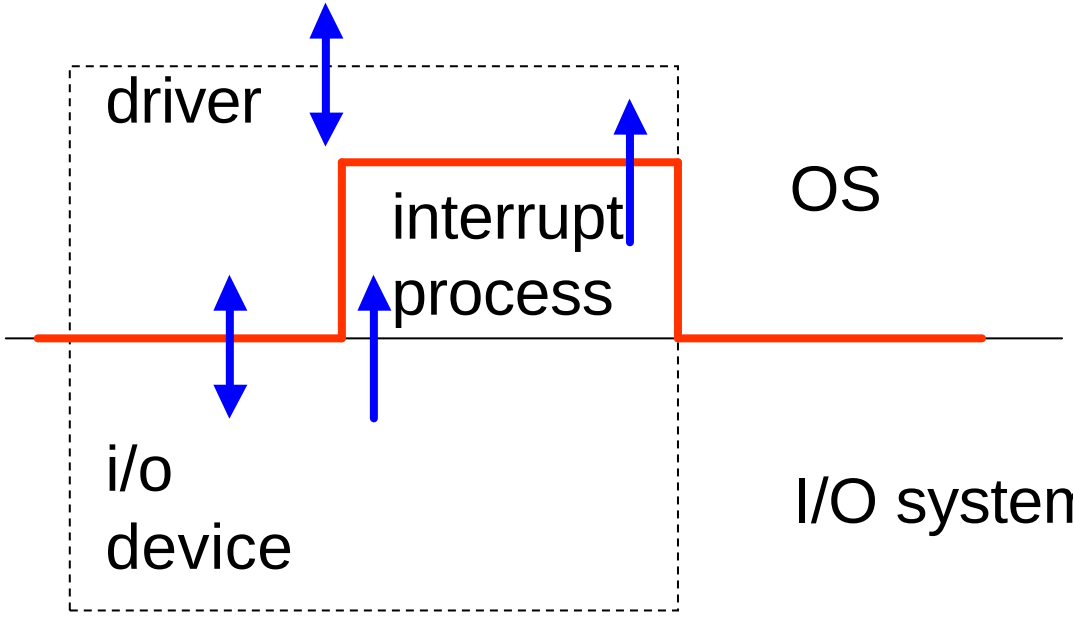
(b)

a) Code executed at print system call
b) Interrupt routine

Driver and interrupt process

Interrupt processes plus drivers build lowest SW level for interface periphery.





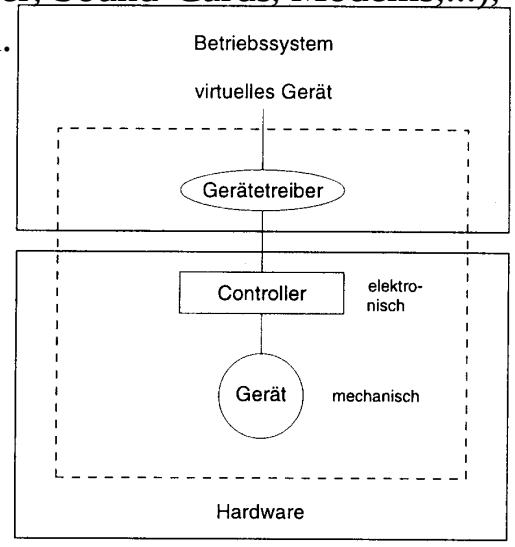
Driver has immediate access to i/o device but all interrupts are masked by the interrupt process (but may be forwarded to the driver).

Driver: part of the SW/HW interface

Aufgabe: Schnittstelle externe HW

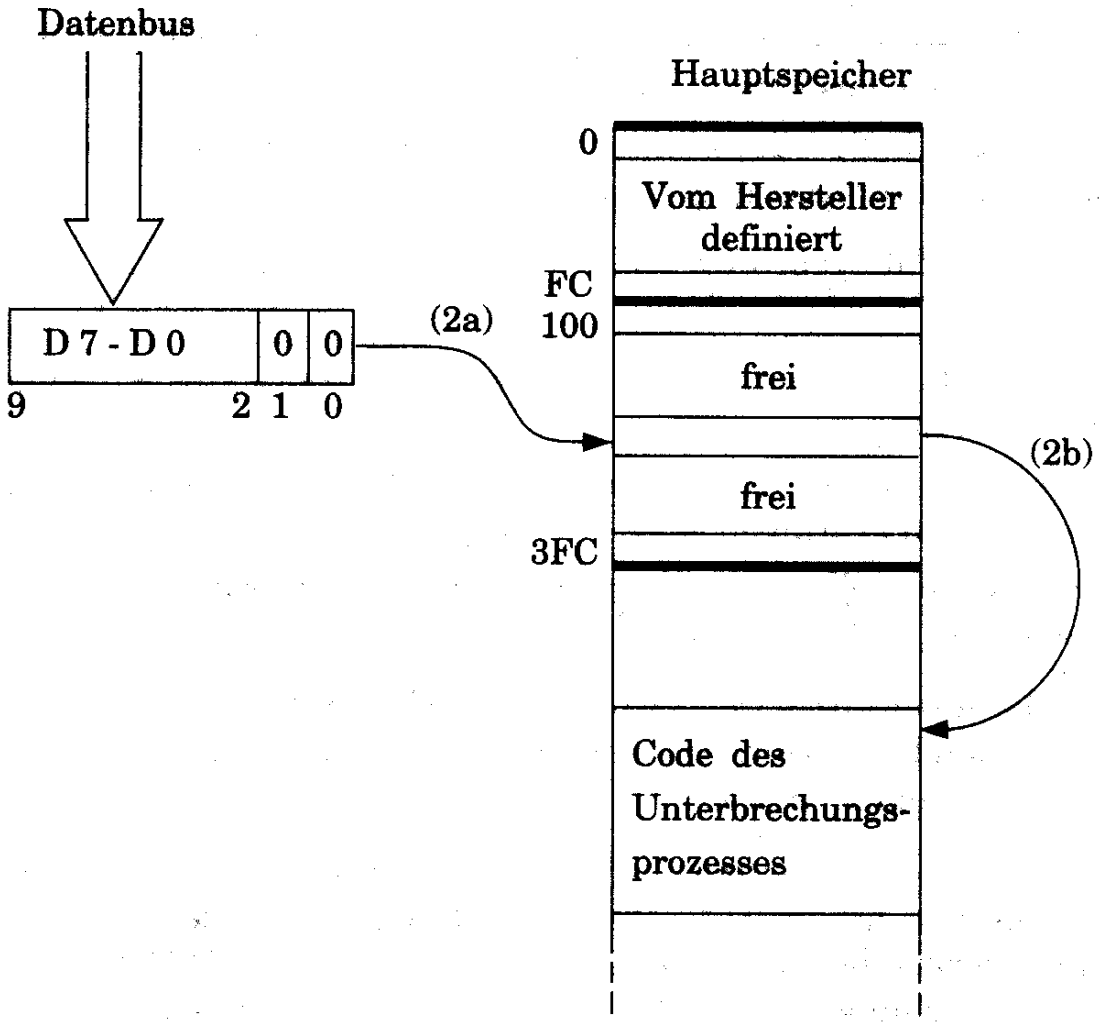
Applikationen sollten auf unterschiedlicher HW mit unterschiedlicher Peripherie laufen (z.B. Drucker, Scanner, Sound-Cards, Modems,...), also (Peripherie-)HW-unabhängig sein. Damit müssen (Peripherie-)HW-abhängige Dienste (s.o.) über das OS ´vermittelt´ werden.

Stichwort: Treiber.

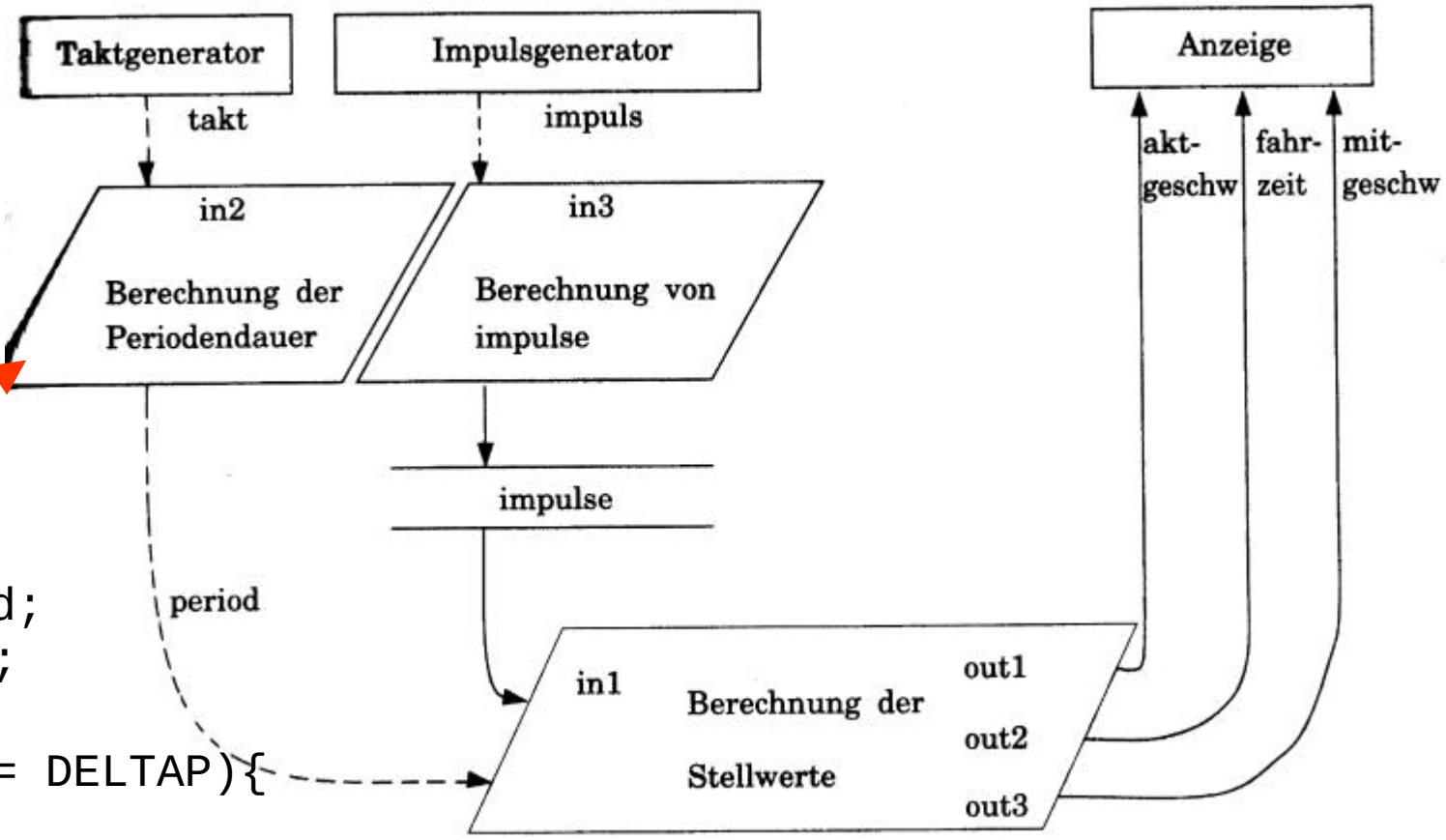


BSS1 1.8

Interrupt handling at the MC 68040



Old example (speedometer) completely with interrupt handling




```
extern int period;
static int j = 0;
in2()
{ if (++j * tG >= DELTAP){
    j = 0;
    wakeup (period);
    }
}
```

Old example (speedometer) completely with interrupt handling

```
#define TRUE 1
#define UMFGANG 2.703
#define DIMKMH 2600/1000
extern int period;
main()
{ int i = 0;          /* Zähler für die Takte einer Periode */
  int fahrtzeit = 0;        /* Fahrtzeit im Periodenraster */
  out vorimpuls = 0;
   while (TRUE) {
       wait (period);
       fahrtzeit = ++i * DELTAP;
       in1 (&impulse);
       aktstrecke = (impulse - vorimpulse) * UMFANG;
       gesstrecke = impulse * UMFANG;
       vorimpulse = impulse;
       out1 (aktstrecke / DELTAP * DIMKMH);
       out2 (fahrtzeit);
       out3 (gesstrecke / fahrtzeit * DIMKMH);
   }
}
```

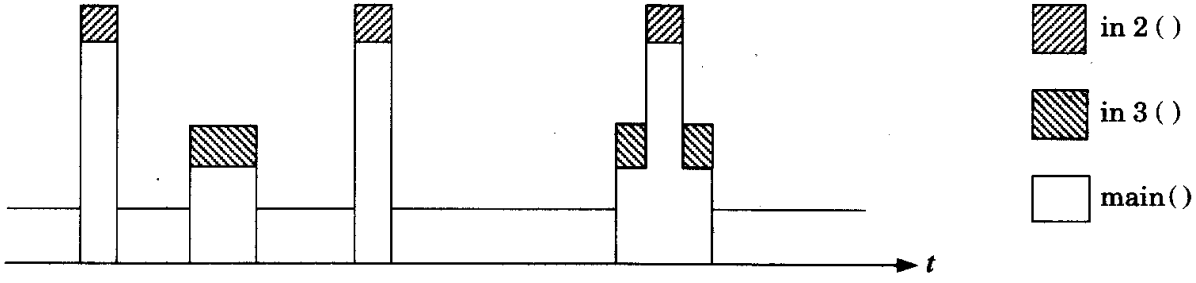
By `wait` triggered calculation of `out1`, `out2`, `out3`

Old example (speedometer) completely with interrupt handling

Priorities:

```
main
in3()   /* calculation of impulses */
in2()   /* calculation of period */
```

```
/* waiting in main */
wait (int x)
{ while (!x); x= 0; }
```



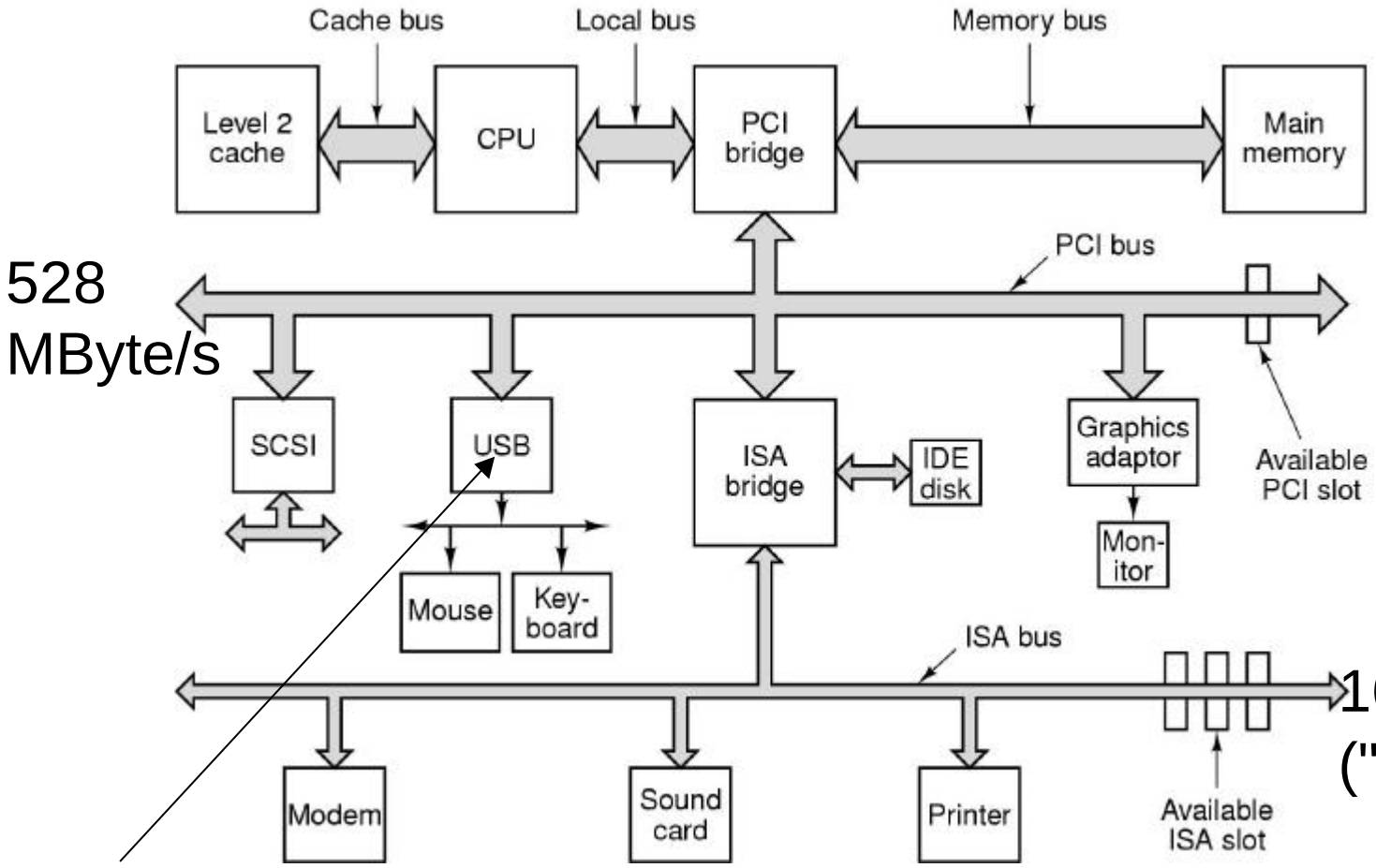

```
/* common variable triggering and triggered process */
wakeup (int x)
{ x = 1; }
```

Internal busses

Also "parallel bus":

- about 100 lines (for different functional groups)
  - address
  - data
  - control
    - bus access
    - interrupt
    - modus
    - errors
  - supply
- many data lines (e.g. 16, 24, 32, 64) for high data throughput
- dimension: 1 (physical) frame, device, less than 1 m
- frame and connections define a standard (plus functionality: protocol)
- examples VME (IEEE 821, IEEE 1101, IEEE 1014), ISA, PCI, SCSI

All these busses



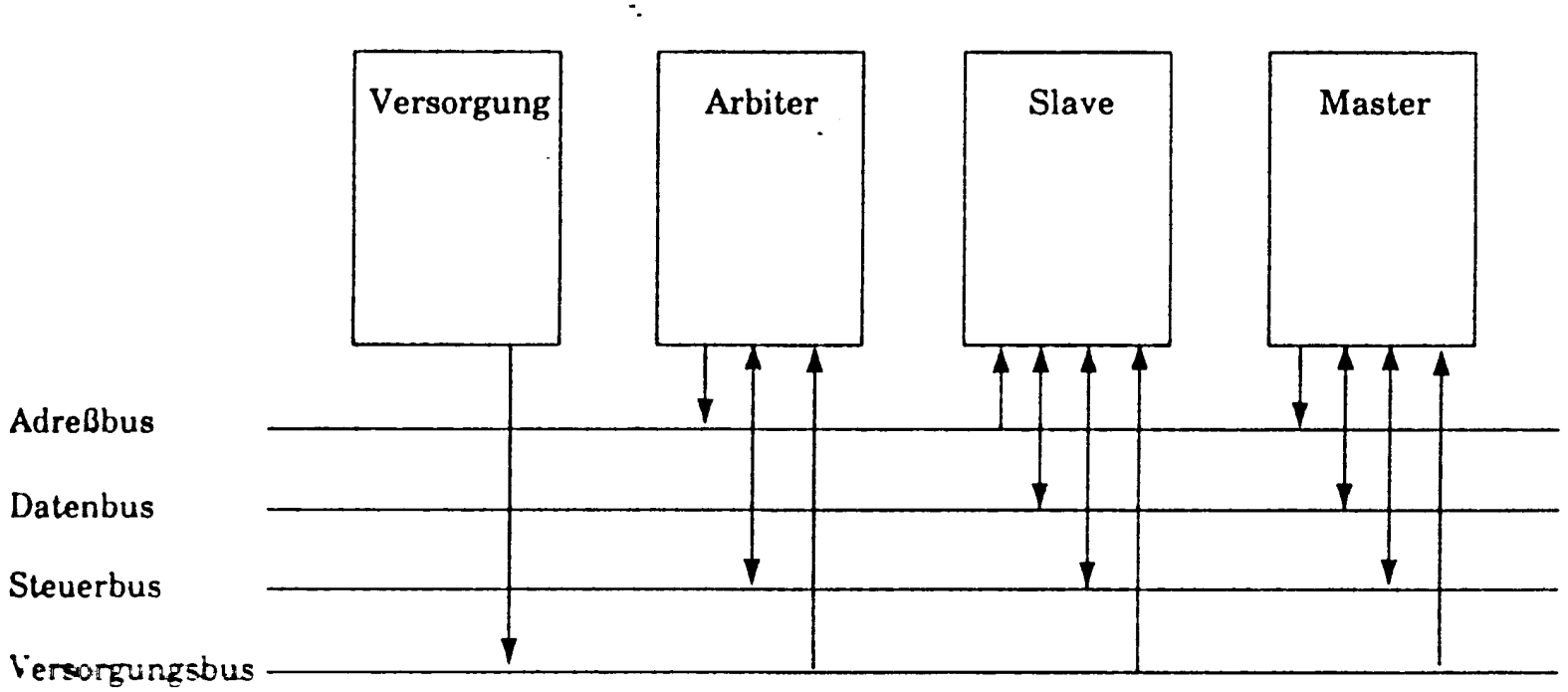
528 MByte/s

16.67 MByte/s ("old" PC/AT bus)

serial!

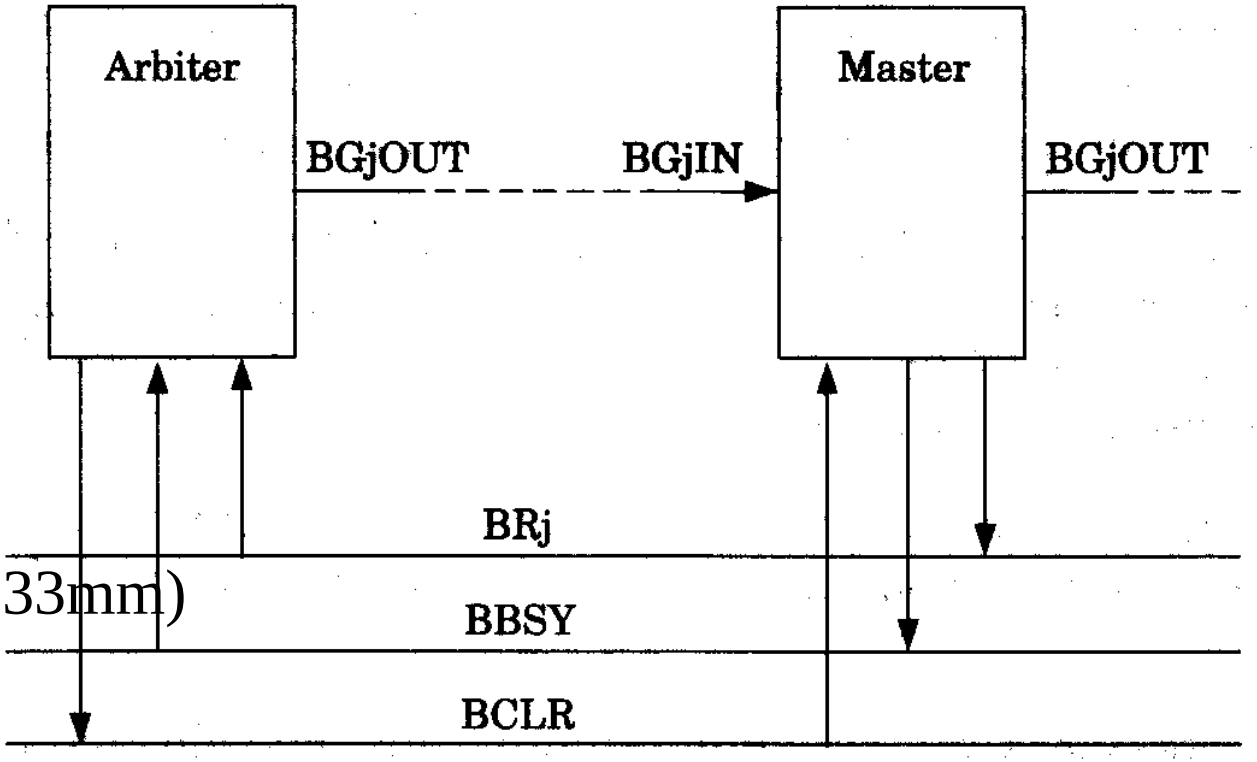
Another one: IEEE 1394 (FireWire), also serial (50 MByte/s)

Arbitration



Modules for (power) supply, arbitration and master and slave each

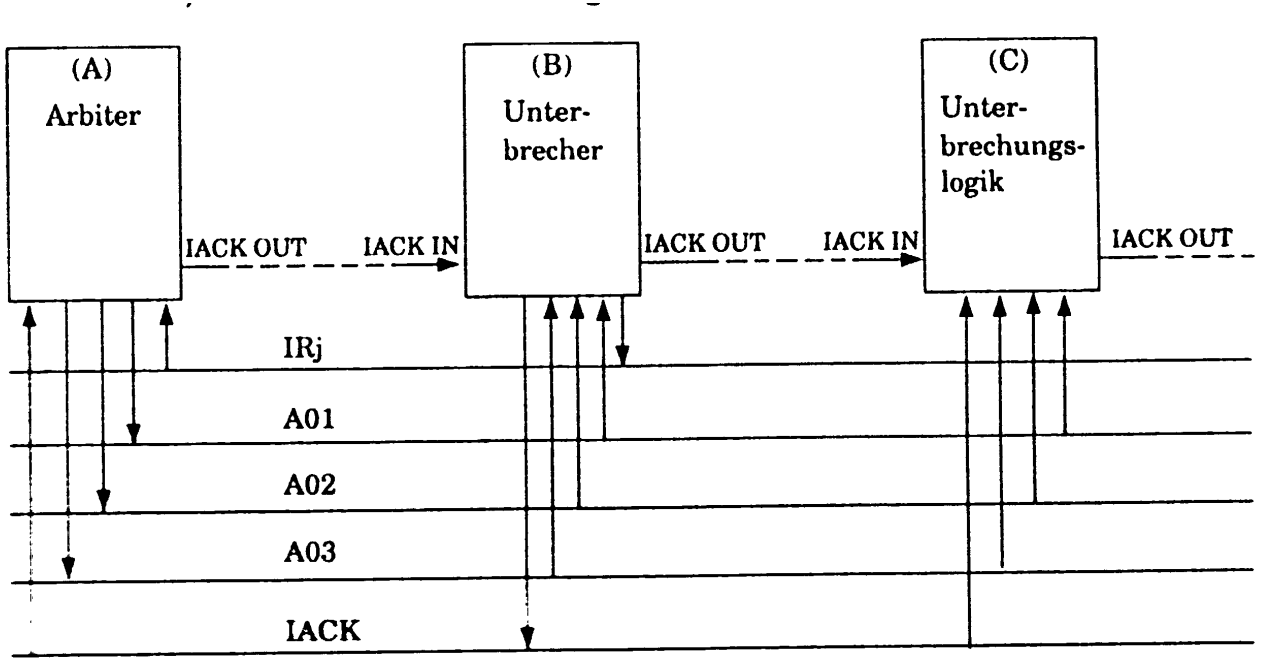
Example: VME (Versa Module Eurocard)



VME:

- double europe (160mm*233mm)
- DIN-603-2 connections
- 64 lines data
- 32 lines address
- 80 MByte/s
- master/slave with arbitration
- asynchronous

Interrupt Acknowledge Cycle

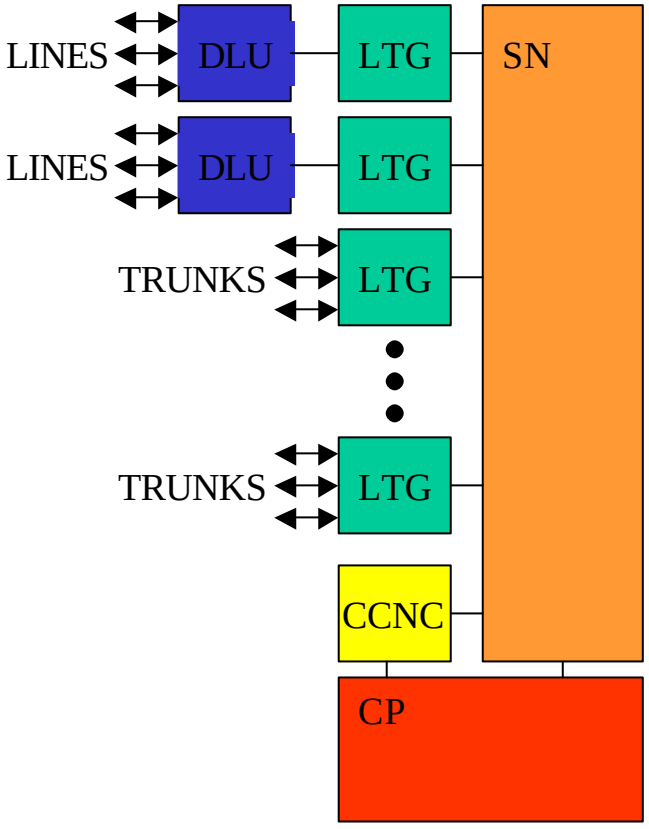


Decisions for an interrupt request

Multi processor systems: bob builder

Two bricklayers need 2 hours for one square meter (brick) wall.
A house is built with 50m² walls.
How long does one bricklayer need for the house, how long need 40 bricklayers?

Example for a multi processor system: SPC (stored program control)

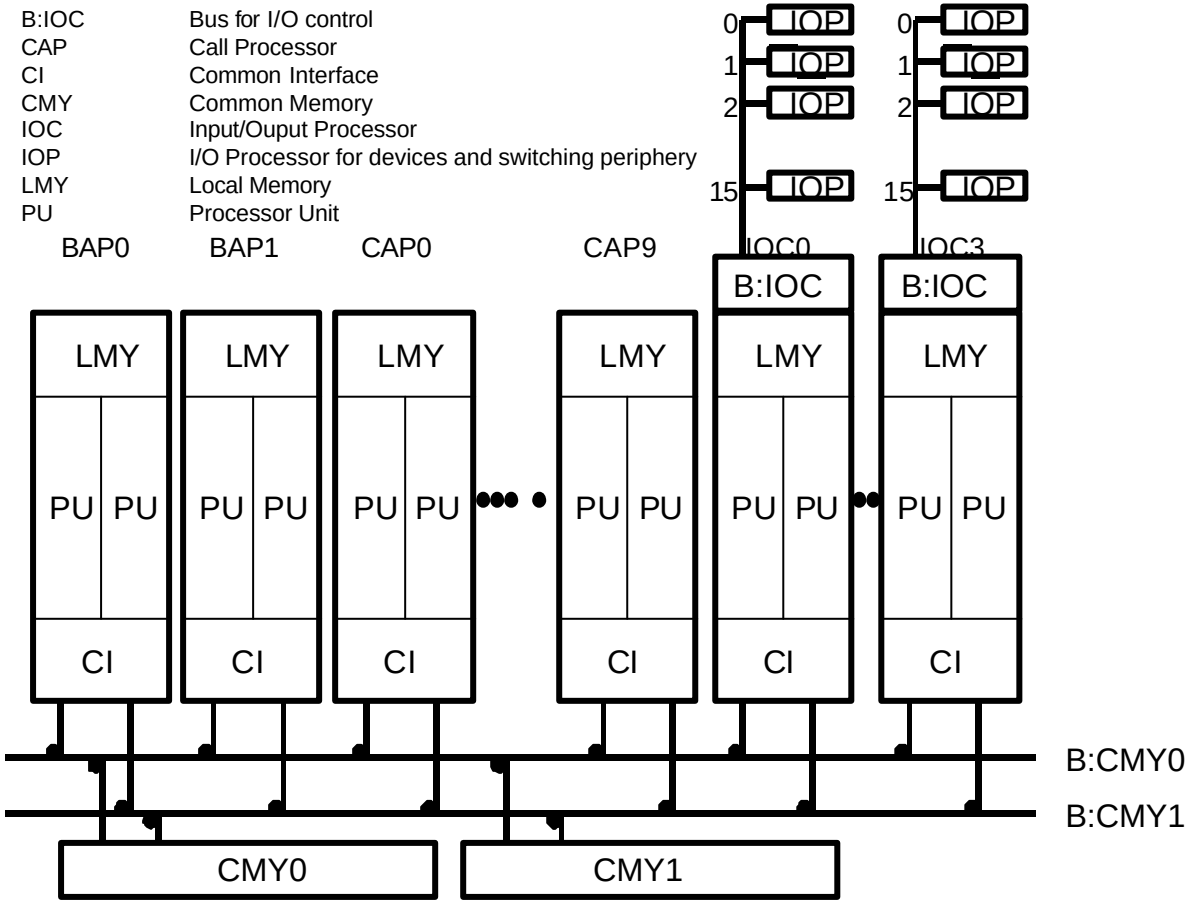


CCNC: Common Channel Signaling Network Control
CP: Coordination Processor
DLU: Digital Line Unit
LTG: Line Trunk Group
SN: Switching Net

/Deinzer99/

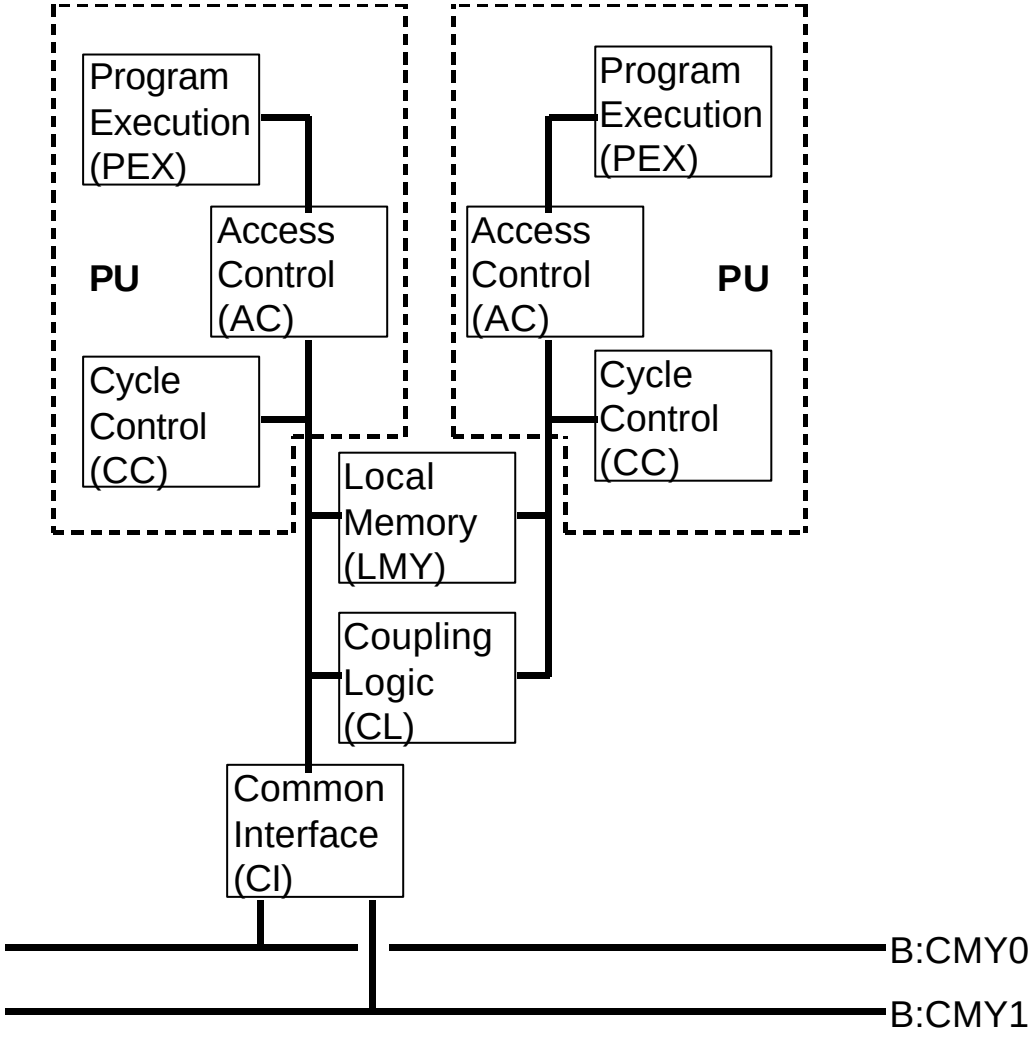
The example goes on: CP as part of SPC

BAP	Base Processor
B:CMY	Bus interface for CMY
B:IOC	Bus for I/O control
CAP	Call Processor
CI	Common Interface
CMY	Common Memory
IOC	Input/Ouput Processor
IOP	I/O Processor for devices and switching periphery
LMY	Local Memory
PU	Processor Unit



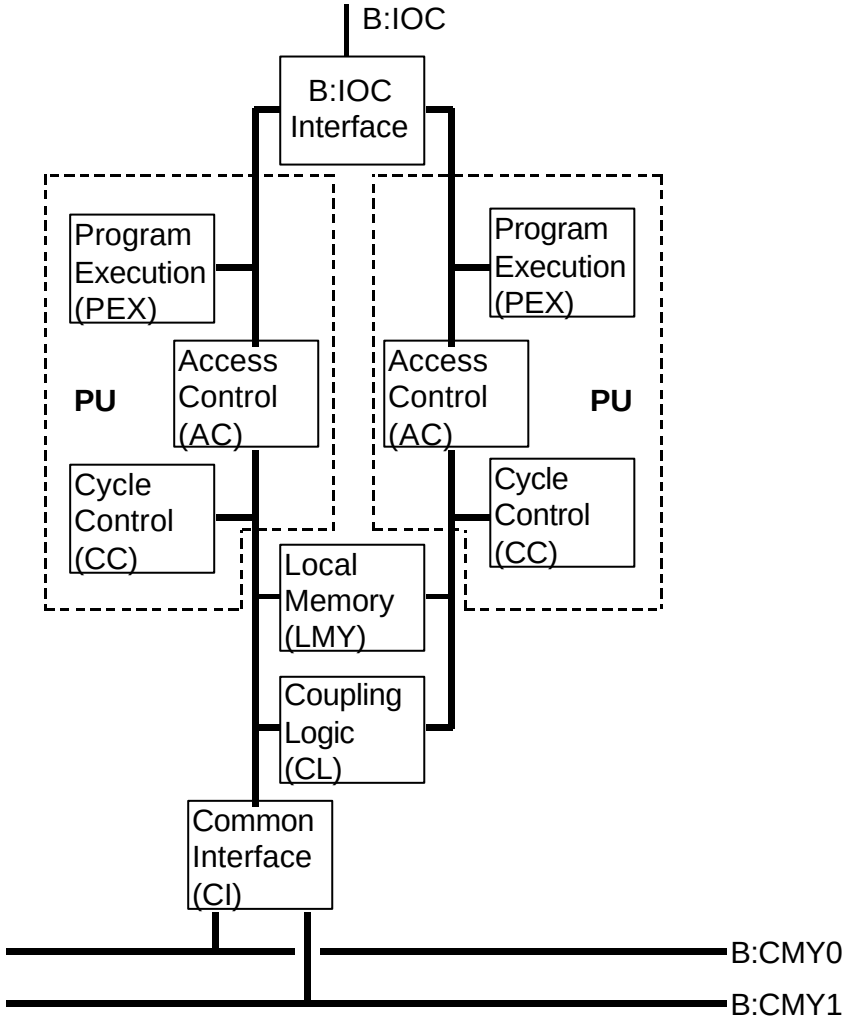
/Deinzer99/

The example goes on: BAP/CAP as part of CP



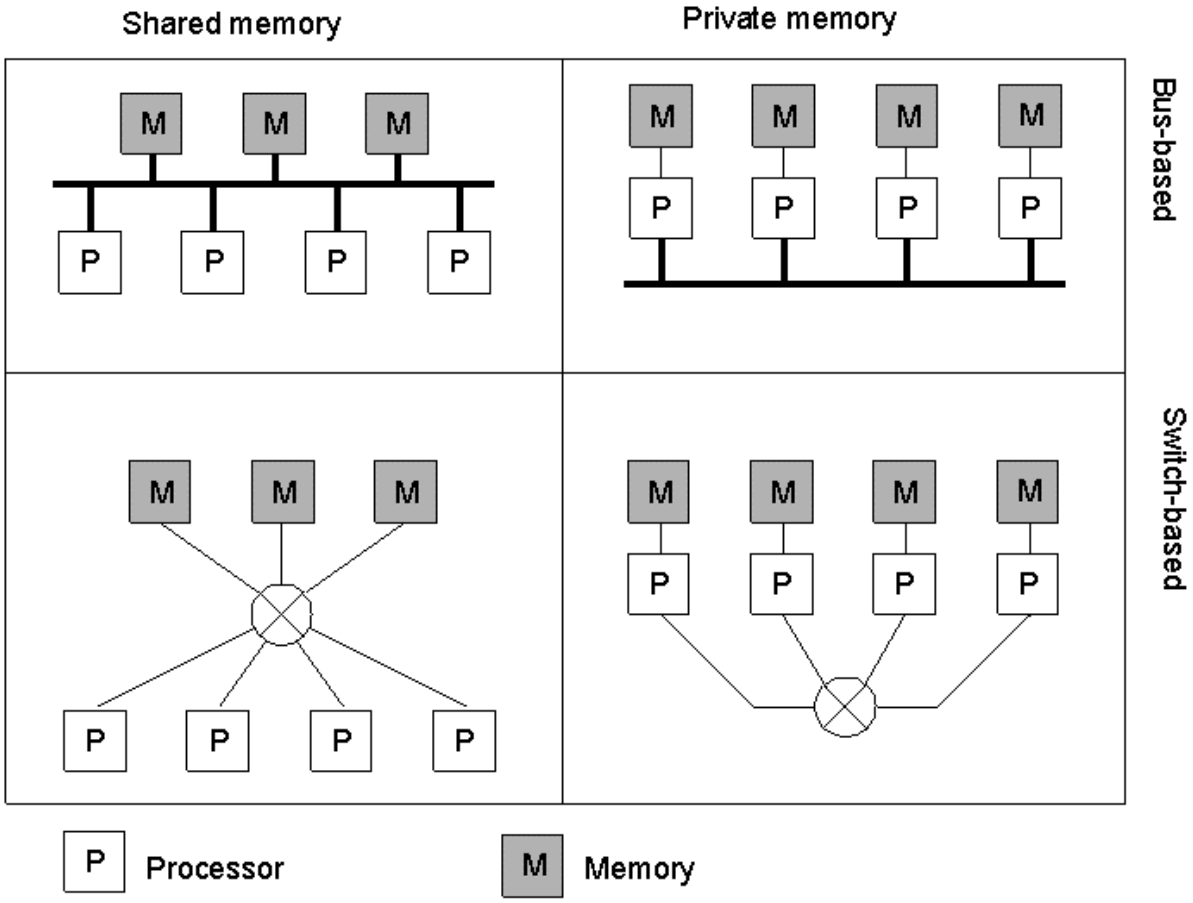
/Deinzer99/

The example goes on: IOP as part of CP



/Deinzer99/

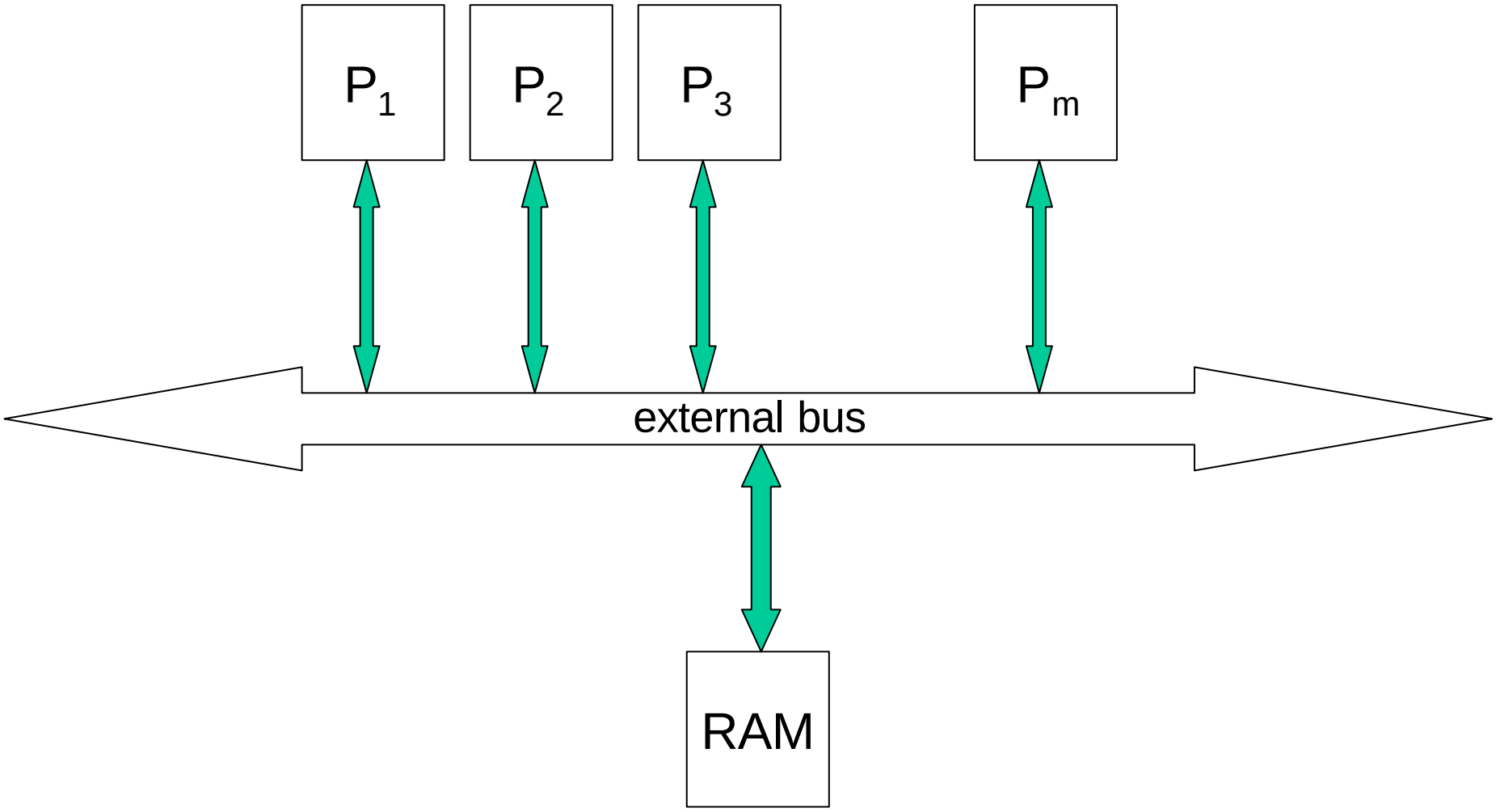
HW concepts for multi processors



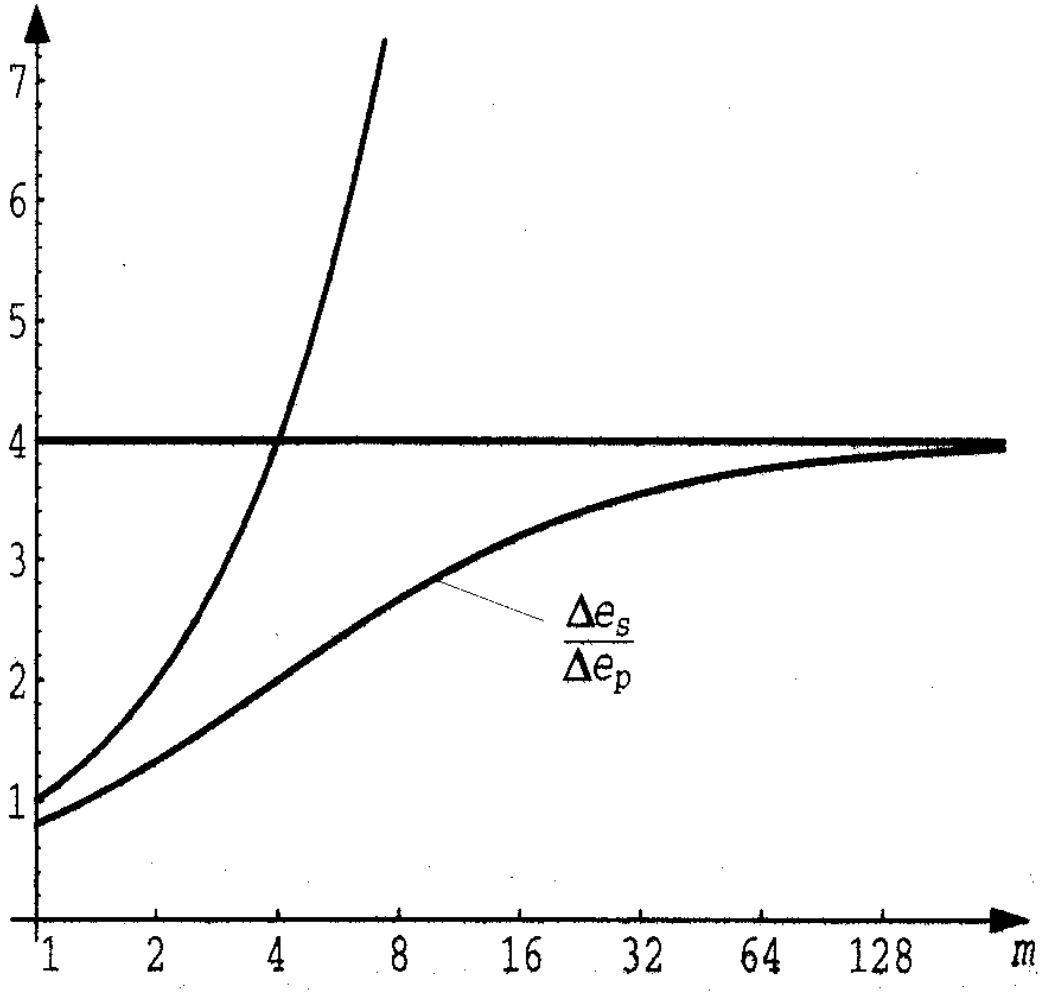
Different basic organizations and memories in distributed computer systems

/Tanenbaum03/

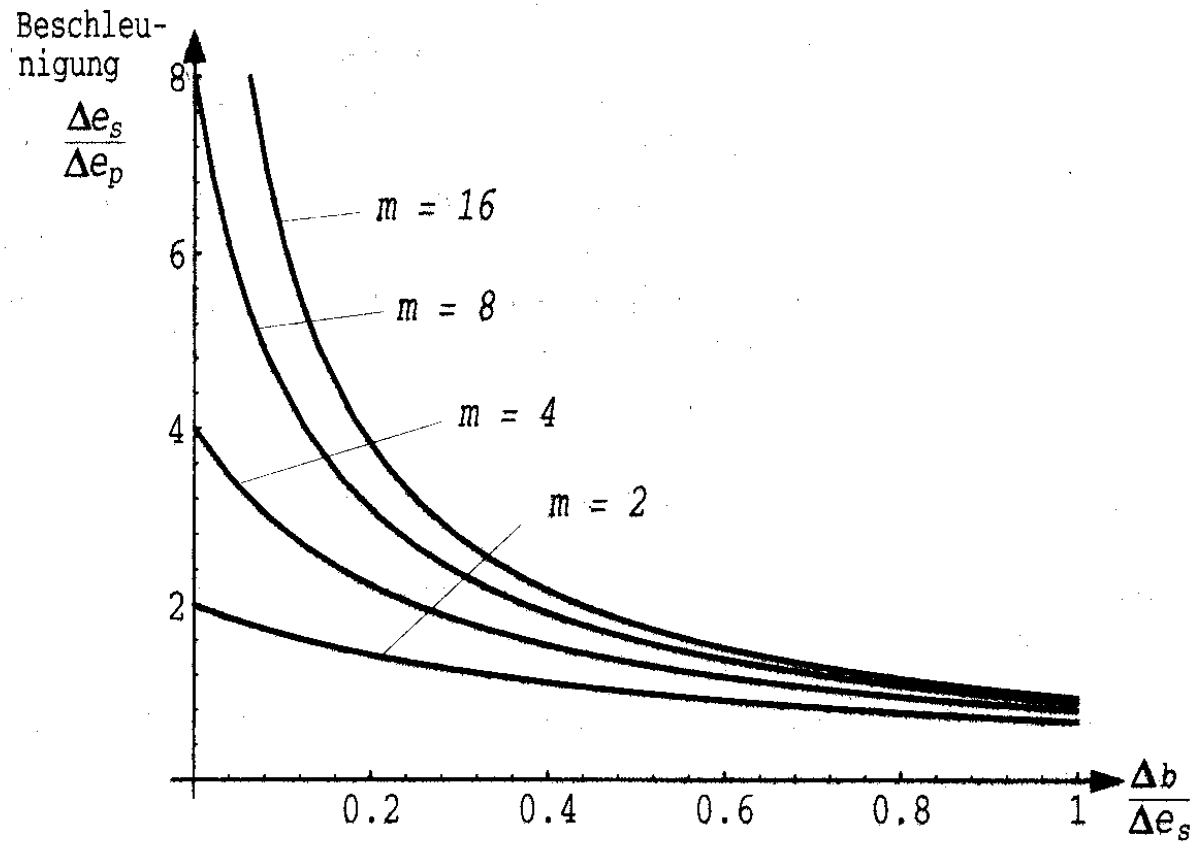
Most simple multiprocessor: shared memory



Shared memory multiprocessor: speed up not linear!

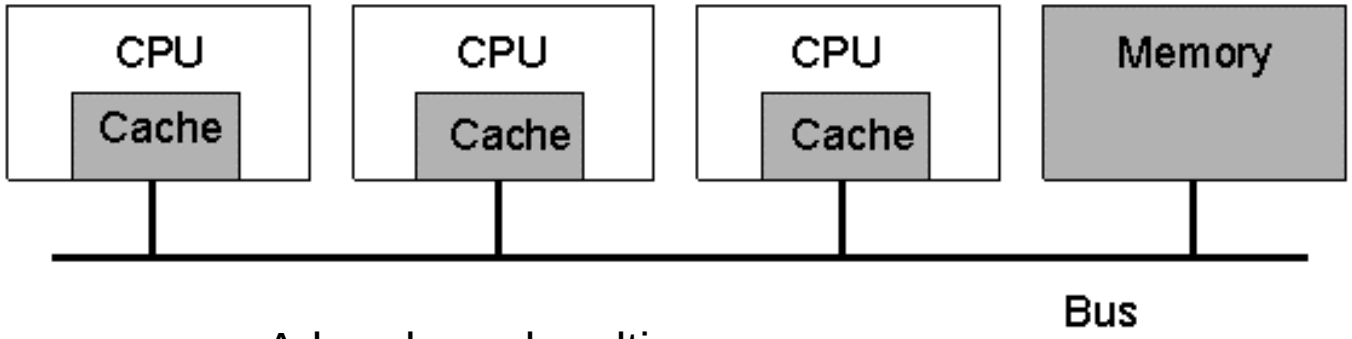


Shared memory multiprocessor: speed up dependent on bus waiting



HW concepts for multi processors - see CP113!

Bus waiting reduced by own memory (cache):



A bus-based multiprocessor.

'Tanenbaum03/

Cache reduces load on external bus, but external bus remains bottle neck:

- load into cache
- store back into memory
- cache coherency (identical data in all caches and memory)

needs still bus.

Appendix: Pseudo Code

following...

Wie programmieren – nur einfacher (Konstanten, aufzählbare Typen)

Vereinbarung von Konstanten:

CONST constname = wert;
Bsp.: CONST MAXINDEX_C = 100;

Vereinbarung von Objektmengen (Datentypen):

TYPE typname = typdefinition;
Bsp.: TYPE FARBE_T = (weiss, schwarz, rot, gelb, gruen);
TYPE STUDIENFACH_T = (Mathematik, Physik, Informatik, Softwareentwicklung);

Wie programmieren – nur einfacher (neue Typen)

Verfügbare Standardtypen: **INTEGER**, **REAL**, **CHAR**, **BOOLEAN**
Unterbereichstypen für aufzählbare Typen:
TYPE typname = [ug..og];
Bsp.: TYPE KLEINBUCHSTABEN_T = ['a'..'z'];
TYPE INDEX_T = [1..MAXINDEX_C];
TYPE AMPELFARBE_T = [rot..gruen];

Felder, Records, Pointer

```
TYPE typname = ARRAY [indextyp] OF typname;
TYPE typname = RECORD
                    Komponentenname: typname;
                    ...
                    END;
TYPE typname = SET OF typname;
TYPE zeigertyp = ^grundtyp;
```

Unterprogramme, Schleifen

```
PROGRAM Schachspiel;
DATA
TYPE SPALTE_T = (A, B, C, D, E, F, G, H);
TYPE ZEILE_T = [1..8];
TYPE AUF_FELD_T =
(leer, Bauer_W, Laeufer_W, Springer_W, Turm_W, Dame_W, Koenig_W,
      Bauer_S, Laeufer_S, Springer_S, Turm_S, Dame_S, Koenig_S);
TYPE SCHACHBRETT_T = ARRAY [SPALTE_T;ZEILE_T] OF AUF_FELD_T;
VARIABLE    Brett: SCHACHBRETT_T;

PROCEDURE Anfang();
{Initialisiert Schachspiel}
DATA
VARIABLE    Zeile: ZEILE_T;
            Spalte: SPALTE_T;
BEGIN
        FOR Spalte:=A TO H DO
                BEGIN
                Brett(Spalte, 2):=Bauer_W;
                Brett(Spalte, 7):=Bauer_S;
                FOR Zeile:=3 TO 6 DO
                        Brett(Spalte, Zeile):=leer;
                END; {FOR Spalte}
END; {Anfang}

BEGIN
        Anfang();
        {und noch ein paar Prozeduren}
END {Schachspiel}
```

Verkettete Liste

Vereinbarung eines Datenobjektes eines gewissen Typs:

VARIABLE variablenname: typname;
Bsp.: VARIABLE ampelzustand: AMPELFARBE_T;

```
CONST   NAMEN_LAENGE_C = 20;
                MAX_ALTER_C = 100;
TYPE            NAMEN_INDEX_T = [0..NAMEN_LAENGE_C-1];
NAMEN_T = ARRAY [NAMEN_INDEX_T] OF CHAR;
ALTER_T = [0..MAX_ALTER_C];
PERSON_P = ^PERSON_T;
                PERSON_T = RECORD
                                        Vorname: NAMEN_T;
                                        Nachname: NAMEN_T;
                                        Alter: ALTER_T;
                                        Next: PERSON_P;
                                END;

VARIABLE Schueler: PERSON_T;

Schueler.Vorname := 'Hugo';
Schueler.Alter := 16;
/*Was darf Hugo ab jetzt?*/
```

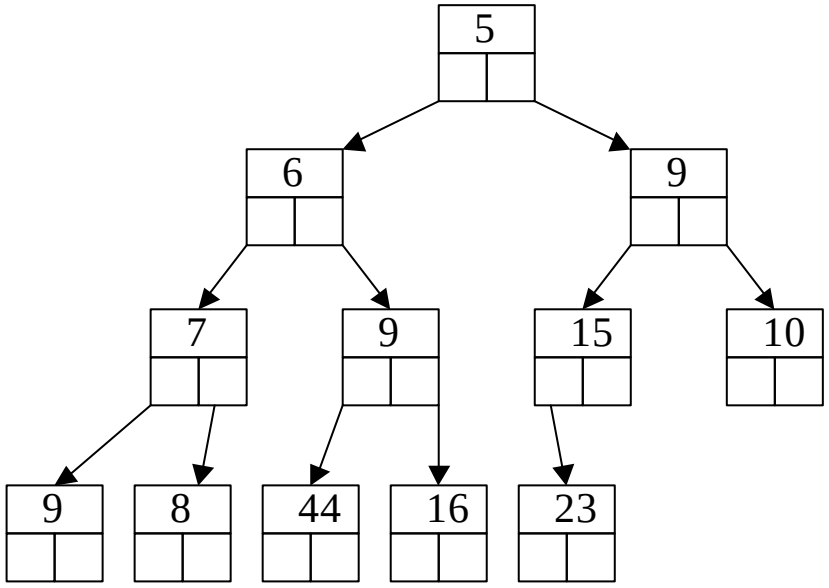
Beispiel: Heapsort

```
PROCEDURE Heapsort;
VAR i : INTEGER;
BEGIN
      HeapAufbauen;
      FOR i:=n DOWNTO 1
      DO BEGIN
            Entnehme HeapSpitze und Speichere als Element (i);
            Kopiere Element unten rechts in HeapSpitze;
            Verkleinere Heap um eins;
            Absenken(HeapSpitze);
      END;{FOR i:=n DOWNTO 1}
END;{Heapsort}
```

Jetzt müssen wir nur noch einen korrekten Heap aufbauen. Auch ganz leicht:

```
PROCEDURE HeapAufbauen;
VAR i : HeapIndex;
BEGIN
      FOR i:=VaterVon(LetztesHeapElement) DOWNTO HeapSpitze
      DO Absenken(i);
END;{HeapAufbauen}
```

Heapsort(1)

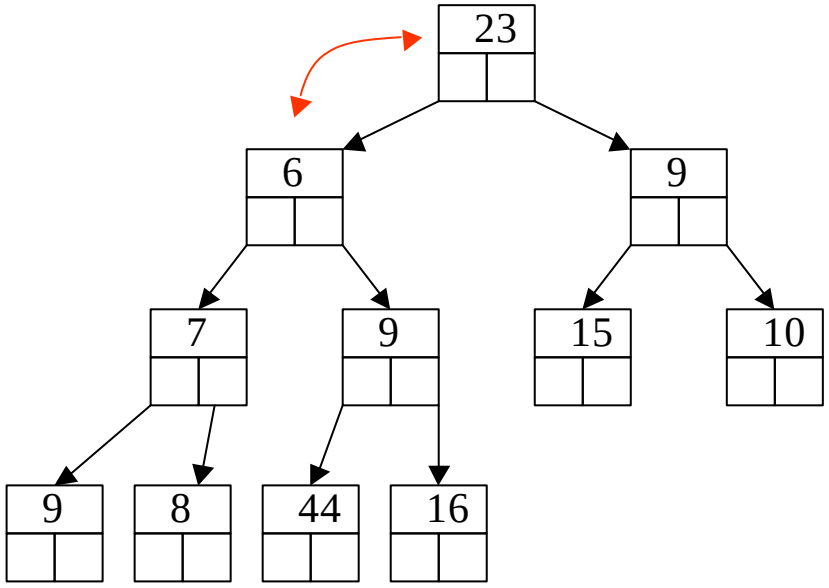


Heapsort(1):

•an der Spitze findet sich das kleinste Element.
•nachdem wir dieses entnommen und abgelegt haben,
•ersetzen wir es duch das letzte Element in der untersten Reihe des Heaps und
•entfernen dieses aus dem Heap.

Damit erhalten wir:

Heapsort(2)



Heapsort(2):

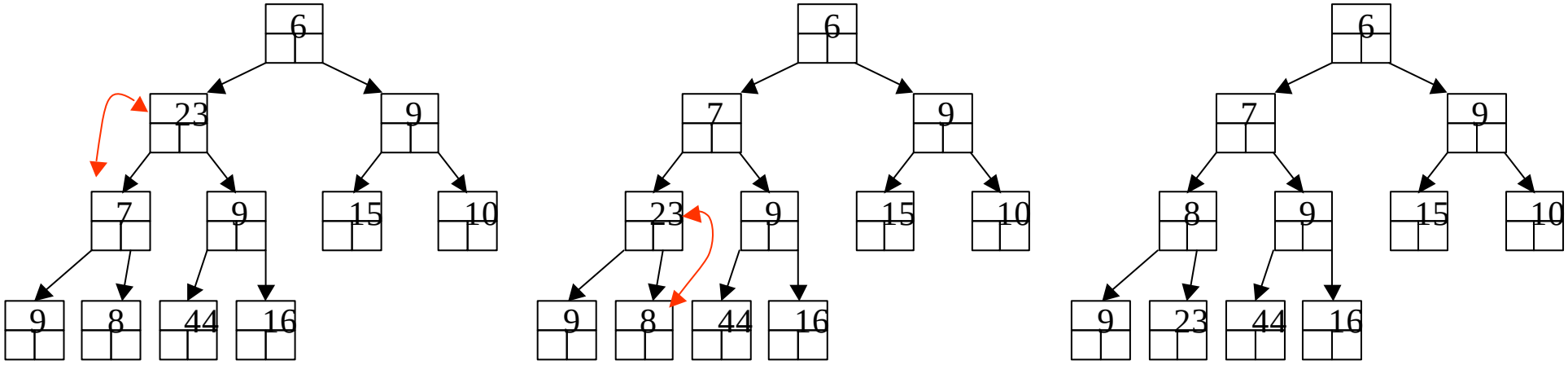
Dies wiederspricht aber der Definition eines Heaps, da 23 nicht kleiner als 6 und 9 ist.

•Wir tauschen daher das unzulässige Element im Heap gegen den jeweils kleinsten Sohn aus.

Diese Prozedur kann sich mehrfach wiederholen, bis das unzulässige Element im Heap soweit ´abgesunken´ ist, dass es nicht mehr den Heap-Regeln widerspricht.

Anschaulich bezeichnet man diese Teilprozedur als **Absenken**. Formulieren Sie **Absenken** in Pseudocode!

Heapsort(3)



Heapsort(3):

Wird die Prozedur **Absenken** mit der Heap-Spitze aufgerufen, ordnet sie eine unzulässige Heap-Spitze innerhalb von wenigen (wie vielen maximal?) Operationen korrekt ein, so dass wieder ein Heap entsteht. Dies setzt aber voraus, dass der Heap zuvor korrekt war mit der einen möglichen Ausnahme der Heap-Spitze.

Im Folgenden werden die Heap-Elemente mit der ´natürlichen´ Indizierung (vgl. Folie ´ Binärer Baum (schichtenweise sequentielle Darstellung)´) angesprochen.

Another example (RealtimeSystems02) (1)

```
PROGRAM EDF_preemptive;
DATA

CONST
MAX_PROCESS_C = 1000;{OS handles up to 1000 processes}
MAX_TIME_C = 1000;{time left in #tG}

TYPE PROCESS_T = RECORD
    NAME: [0..MAX_PROCESS_C];
    DEADLINE: [1..MAX_TIME_C];
    EXTIME: [1..MAX_TIME_C];{remaining!}
END;{RECORD}

TYPE PROCESS_QUEUE_ELEMENT_T = RECORD
    PROCESS: PROCESS_T;
    NEXT: PROCESS_QUEUE_ELEMENT_P;
END;{RECORD}

TYPE PROCESS_QUEUE_ELEMENT_P=^PROCESS_QUEUE_ELEMENT_T ;

VARIABLE
EL PROCESS_QUEUE_ELEMENT_P;{Pointer to ED list};
p,newp PROCESS_T;
```

Another example (RealtimeSystems02) (2)

```
PROCEDURE Init;
{Initialises EL}
BEGIN
   EL=new(PROCESS_QUEUE_ELEMENT_P);
   EL^.NAME=0;{Idle Process}
   EL^.DEADLINE=MAX_TIME_C;{runs only if no other process available}
END; {Init}

PROCEDURE head(Queue PROCESS_QUEUE_ELEMENT_P): PROCESS_T;
{Results process with ED}
BEGIN
   RETURN=Queue^.PROCESS;
END; {head}

PROCEDURE insert(Element PROCESS_T;
Queue PROCESS_QUEUE_ELEMENT_P);
{Sorts process into queue ordered by deadline}
BEGIN
   {…};
END; {head}
```

Another example (RealtimeSystems02) (3)

```
PROCEDURE delete(Element PROCESS_T;
Queue PROCESS_QUEUE_ELEMENT_P);
{Removes process from queue}
BEGIN
    {…};
END; {delete}
```

Another example (RealtimeSystems02) (4)

```
BEGIN {main}
Init;
WHILE TRUE DO
    p:=head(EL);
    dispatch(p);{p gets CPU, extime is reduced}
    IF ARRIVAL(newp)<TIME(rest(p))
       THEN insert(newp,EL){and scheduling begins again}
       ELSE delete(p,EL); {p is ready and is removed from queue}
END {EDF_preemptive}
```